Kolmogorov.ai | Continuity

Decision Ops

ООО Дата Сапиенс

v.1.1.0 | 1.11.2023

Table of contents

1. Continuity.

Continuity [k nt nju( ) ti] - инструмент класса Decision Ops. Предназначен для операционализации моделей.

Обеспечивает поддержку жизненного цикла систем анализа данных в бизнес-процессах различных индустрий.

Continuity A2P (Advanced Analytics Platform) - модуль, состоящий из Open Source инструментов, предназначенный для увеличения прозрачности и скорости внедрения ML-моделей в промышленную эксплуатацию.

Скачать документацию в формате PDF

2. Руководство по установке

2.1 Введение

2.1.1 Установка

Первичная установка программного обеспечения производится командой разработки и внедрения ООО «Дата Сапиенс» на сервер, предоставляемый заказчиком.

2.1.2 Обновление

Проверка, загрузка и установка обновлений программного обеспечения требует подключения к Интернету и выполняется вручную на сервере администратором ПО. Для обновления комонента ПО необходимо удалить его и установить повторно, после чего снова запустить Kolmogorov.ai **Continuity**.

Любую доступную версию Kolmogorov.ai **Continuity** можно загрузить и установить с помощью контейнеров, поставляемых разработчиком. Перед переустановкой допускается удаление существующего ПО.

2.2 Пререквизиты

2.2.1 Kubernetes

Должен быть развернуть кластер Kubernetes и установлен инструмент для управления Kubernetes приложениями Helm

Установка Kubernetes на minikube

Рекомендации от команды DevOps

2.2.2 Keycloak

Сервис для авторизации и аутентификации Keycloak.

Рекомендуется использовать чарт codecentric

2.2.3 Minio

S3 хранилище

Инструкция по установке Minio

2.3 Сервер

2.3.1 Серверная часть

Kolmogorov.ai **Continuity** - набор инструментов для операционализации бизнес-решений.

Основной сервис бекенда - FastAPI сервер, который принимает запросы по адресу, описанному в ingress сервиса. Часть задач выполняться асинхронно с использованием Celery. Camunda используется для управления бизнес-сценариями, являющимися workflow проекта, внешние задачи Camunda выполняются специальными workers. Метаданные проектов хранятся в БД Postgres.

2.3.2 Устройство чарта

Чарт содержит в себе следующие сервисы:

Service

Основной сервис бекенда. Внутри контейнера разворачивается FastAPI сервер, которые принимает запросы по адресу, описанному в ingress

Содержит init-container `init-db`, применяющий миграции Alembic к базе данных, URL которой указан в переменной `CONTINUITY_DB_URL`.

Внутри этого сервиса в values.yaml описываются переменные окружения, которые потом собираются в секрет `continuity-backend-secret`.

Postgresql

База данных на базе Postgres для

Service-celery

Сервис запускает воркер для Celery

Redis

Брокер сообщений для celery задач

2.3.3 Переменные окружения

Название	Описание	Тип данных	Значение по умолчанию
TZ	Часовой пояс	str	Europe/Moscow
CONTINUITY_ADMIN_ROLE	Название роли администратора	str	
CONTINUITY_AUDIT_ROLE	Название роли наблюдения	str	
CONTINUITY_ROOT_PATH	root_path для сервера Continuity на базе FastAPI	str	/api
CONTINUITY_DB_URL	Адрес Postgres базы данных в формате "postgresql://:@:/" В качестве рекомендуется использовать название сервиса базы Continuity-postgresql	str	postgresql://:@Continuity-postgresql:5432/postgres
CONTINUITY_DB_POOL_SIZE	Количество одновременных подключений к базе	int	30
CONTINUITY_DB_CONNECT_TIMEOUT	Время ожидания подключения к базе данных(в секундах)	int	10
CONTINUITY_KEYCLOAK_USERNAME	Логин администратора Keycloak	str	
CONTINUITY_KEYCLOAK_PWD	Пароль пользователя Keycloak	str	
CONTINUITY_KEYCLOAK_URL	Адрес Keycloak для получения токена в формате https:/// auth/realms// protocol/openid-connect/token	str	
CONTINUITY_KEYCLOAK_SERVER_URL	Адрес Keycloak для авторизации в формате https:///admin	str	
CONTINUITY_KEYCLOAK_REALM	Название realm Keycloak	str	
CONTINUITY_KEYCLOAK_CLIENT	Название приложения(client) Keycloak	str	
CONTINUITY_S3_URL	Адрес S3 хранилища	str	
CONTINUITY_S3_USERNAME	Логин для доступа к S3 хранилищу	str	
CONTINUITY_S3_PWD	Пароль для доступа к S3 хранилищу	str	

Название	Описание	Тип данных	Значение по умолчанию
CONTINUITY_S3_BUCKET	Имя S3 bucket	str	
CONTINUITY_S3_VERIFY	Проверка SSL сертификатов при подключении к S3	bool	False
CONTINUITY_CELERY_NAME	Имя Celery	str	
CONTINUITY_CELERY_STAGE_HEALTH_DELAY	Время между запусками регулярных задач Celery	float	10 * 60
CONTINUITY_CELERY_BROKER_URL	Адрес брокера сообщений Redis в формате redis://:@: В качестве рекомендуется использовать название сервиса continuity-redis-master. Значение по умолчанию 6370	str	
CONTINUITY_CELERY_RESULT_BACKEND	Адрес БД с результатами выполнения Celery задач в формате redis://:@: В качестве рекомендуется использовать название сервиса continuity-redis-master. Значение по умолчанию 6370	str	
CONTINUITY_CELERY_TASK_TABLE_NAME	Имя таблицы тасок Celery. См. документацию	str	celery_taskmeta
CONTINUITY_CELERY_GROUP_TABLE_NAME	Имя таблицы групп Celery. См. документацию	str	celery_group
BPMN_URL	Адрес Camunda	str	
BPMN_USERNAME	Логин пользователя для доступа к Camunda	str	
BPMN_PASSWORD	Пароль пользователя для доступа к Camunda	str	
CONTINUITY_K8S_HOST	Адрес K8S	str	https://kubernetes.default
CONTINUITY_K8S_PATH_TO_TOKEN	Путь к файлу с токеном	str	/var/run/secrets/kubernetes.io/serviceaccount/token
CONTINUITY_K8S_PATH_TO_NAMESPACE	Путь к файлом с namespace	str	

Название	Описание	Тип данных	Значение по умолчанию
			/var/run/secrets/kubernetes.io/serviceaccount/namespace
CONTINUITY_SENTRY_ENABLED	Флаг мониторинга ошибок с помощью Sentry	bool	False
CONTINUITY_SENTRY_DSN	Имя источника данных Sentry	str	None

2.3.4 Установка чарта сервиса Continuity

1. Отредактируйте файл values.yaml
   - Подставьте актуальный репозиторий и тег образа в следующих сервисах:
   - service > image
   - service-celery -> image
   - service-flower -> image
   - Подставьте актуальный host и baseDomain в следующих сервисах
   - service > ingress
   - Задайте секрет, содержащий переменные окружения. Для чувствительных даннных используете Vault
2. Если установка релиза происходит первый раз, то раскомментировать строчку - "--metrics" в service -> init -> containers -> args, чтобы заполнить таблицу Metrics базовыми метриками
3. Запустите команду

```
helm upgrade
--namespace=<namespace> \
--install \
--atomic \
<realisename> <path to chart>
```

где `namespace` - неймспейс k8s, в котором устанавливается релиз, `realisename` - название релиза, `path to chart` - путь до папки чарта или до архива

2.3.5 Обновление

Для обновления чарта запустите команду:

```
helm upgrade
--namespace=<namespace> \
--install \
--atomic \
<realisename> <path to chart>
```

где `namespace` - неймспейс k8s, в котором устанавливается релиз, `realisename` - название релиза, `path to chart` - путь до папки чарта или до архива

2.3.6 Удаление

Для удаления чарта запустите команду:

```
helm delete --namespace=<namespace> <realisename>
```

где `namespace` - неймспейс k8s, в котором устанавливается релиз, `realisename` - название релиза,

2.4 UI

2.4.1 Серверная часть

web-интерфейс Contunity.

2.4.2 Устройство чарта

Чарт содержит в себе один сервис: predicate-frontend с файлами web-приложения. Сервис имеет два init-контейнера:

- generate-configs Настраивает перменные окружения PREDICATE_API и DATASOURCE_REGISTRY_API
- nginx-config Запускает nginx-сервер

2.4.3 Переменные окружения

Название	Описание	Тип данных	Значение по умолчанию
CONTINUITY_API	Адрес API, обычно совпадает с адресом интерфейса с префиксом /api	str	
PREDICATE_HOST	Адрес Predicate	str	
PREDICATE_API	Адрес API Predicate	str	
KEYCLOAK_REALM	Имя realm Keycloak. Обычное совпадает с realm, указанным в бекенде	str	
KEYCLOAK_AUTH_SERVER_URL	Адрес Keycloak для авторизации в формате https:///auth Обычное совпадает с указанным в бекенде	str	
KEYCLOAK_RESOURCE	Название приложения(client) Keycloak	str	
Locale	Часовой пояс	str	Europe/Moscow

2.4.4 Установка чарта сервиса Continuity

1. Отредактируйте файл values.yaml
   - Подставьте актуальный репозиторий и тег образа в следующих сервисах:
   - service > image
   - service-celery -> image
   - service-flower -> image
   - Подставьте актуальный host и baseDomain в следующих сервисах
   - service > ingress
   - Задайте секрет, содержащий переменные окружения. Для чувствительных даннных используете Vault
2. Если установка релиза происходит первый раз, то раскомментировать строчку - "--metrics" в service -> init -> containers -> args, чтобы заполнить таблицу Metrics базовыми метриками
3. Запустите команду

```
helm upgrade
--namespace=<namespace> \
--install \
--atomic \
<realisename> <path to chart>
```

где `namespace` - неймспейс k8s, в котором устанавливается релиз, `realisename` - название релиза, `path to chart` - путь до папки чарта или до архива

2.4.5 Обновление

Для обновления чарта запустите команду:

```
helm upgrade
--namespace=<namespace> \
--install \
--atomic \
<realisename> <path to chart>
```

где `namespace` - неймспейс k8s, в котором устанавливается релиз, `realisename` - название релиза, `path to chart` - путь до папки чарта или до архива

2.4.6 Удаление

Для удаления чарта запустите команду:

```
helm delete --namespace=<namespace> <realisename>
```

где `namespace` - неймспейс k8s, в котором устанавливается релиз, `realisename` - название релиза,

3. Руководство пользователя

3.1 О приложении

3.1.1 Общая информация

Программное обеспечение Kolmogorov.ai Continuity представляет собой набор инструментов для операционализации бизнес-решений.
Обеспечивает поддержку жизненного цикла систем анализа данных в бизнес-процессах различных индустрий.

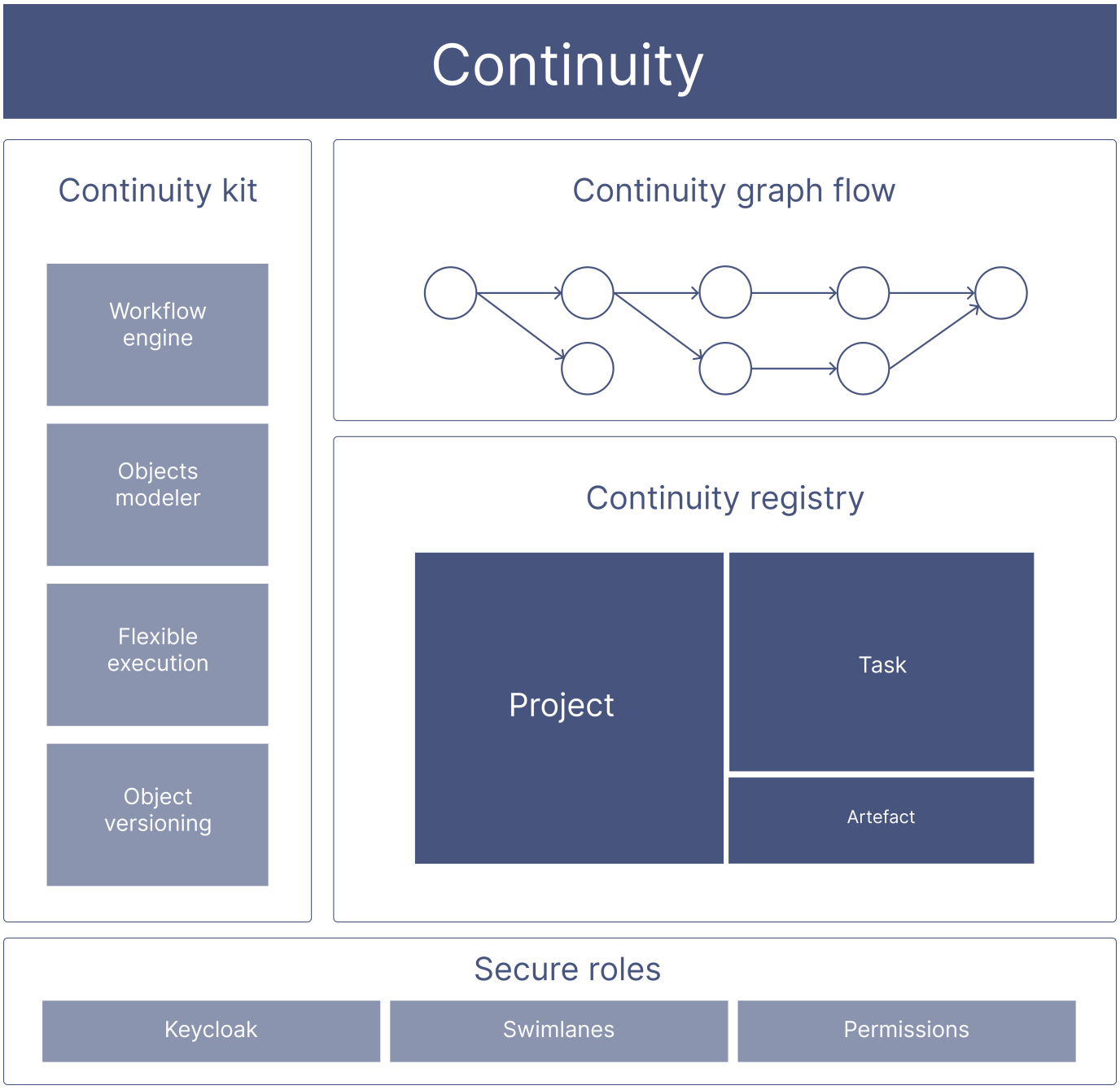
ИЗ ЧЕГО СОСТОИТ KOLMOGOROV.AI CONTINUITY

Continuity workflow.
Представляет собой ориентированный граф, где есть вершины (Nodes) и направленные ребра (Edges). Прохождение бизнес процессов автоматизируется с помощью графа.

Continuity registry.
Хранит созданные в системе объекты.

Continuity kit.
Набор инструментов для управления графами и объектами. Включает в себя:
- Движок автоматизации workflow
- Конструктор объектов
- Функционал по исполнению процесса
- Версионирование объектов.

Ролевая модель.
Предусматривает авторизацию через Keycloak и предоставление прав доступа на все объекты (permission).



Объектная модель Continuity содержит три базовых типа объектов:

1. Проект
2. Задача
3. Артефакт

С помощью базовых объектов можно выстраивать гибкую иерархию любого порядка и вложенности. Объекты в системе можно шаблонизировать с помощью Конструктора.

CONTINUITY WORKFLOW

Процессы Continuity строятся на базе графа, где узлы представляют собой контейнеры для сущностей, а ребра – связи между этими контейнерами.

Проект, задача и артефакт – это типы запущенного пользователем workflow. Пользователь создает объект в runtime из заранее настроенного шаблона. После запуска объекта внутри него создается узлы первой очереди графа. А далее процесс двигается пользователем, согласно настроенным в шаблоне ребрам.

Более подробная информация о простроении графа: Граф Continuity

CONTINUITY REGISTRY

Registry хранит информацию о всех созданных объектах в системе. В Continuity есть 3 каталога для runtime объектов: проекты, задачи, артефакты. Так как объекты в Continuity обычно вложены друг в друга, доступен просмотр всех нижестоящих сущностей в любом объекте, открытом из каталога.

В Continuity также есть каталог шаблонов, который хранит все созданные в системе шаблоны.

ИНСТРУМЕНТЫ CONTINUITY

Шаблоны - это смоделированные пользователем сущности, на основе которых могут быть запущены объекты в runtime.

Шаблон может быть настроен пользователем с помощью конструктора.

Подробнее: Работа с шаблонами

Тегирование

Пользователь может помечать объекты тегами. Это позволяет быстро находить похожие объекты.

Подробнее: Тегирование объектов

Версионирование

Все объекты в Continuity имеют версии. Это помогает сохранить в системе историю изменений.

Подробнее: История изменений объекта

Установка программного обеспечения производится командой разработки и внедрения ООО «Дата Сапиенс» на сервер, предоставляемый заказчиком.

3.1.2 Глоссарий

Объектная модель Continuity (Object model) представляет собой иерархически организованную структуру с гибкой настройкой типов и вложенности объектов.

Объект (Object) - сущность системы Continuity, имеющая уникальный идентификатор набор определенных свойств. Объект может принимать тип проекта, задачи или артефакта. Объект является единицей иерархии, то есть может содержать в себе вложенные объекты либо быть дочерним по отношению к вышестоящей сущности.

Тип объекта (Type) - это определение объекта, регламентирующее его вид и правила взаимодействия с ним. В системе есть 3 типа объекта: Проект, Задача, Артефакт.

Граф Continuity (Graph) - это механизм для автоматизации бизнес-процессов. У него есть ребра, которые определяют направление движения workflow, и ноды, являющиеся контейнерами для объектов Continuity.

Подробнее о графе: Граф Continuity

Нода (Node) - это элемент графа и контейнер для объекта Continuity. Необходим для корректной работы автоматизации процессов.

Ребро графа (Edge) обеспечивает связь между узлами. По ребрам графа задается направление workflow.

Шаблон объекта (Template) - это описание объекта, его свойств, внутренней иерархии. Шаблон позволяет переиспользовать объекты более одного раза. Пользователю доступно настройка шаблона через Конструктор.

Подробнее о шаблоне: Шаблон объекта

Контекст объекта (Context) - свойство объекта/узла/ребра. С помощью контекстов можно настраивать тип объекта, его отображение, условия перехода в рамках workflow.

Подробнее о контекстах: Контекст объекта

Свимлейн (Swimlane) - это инструмент Continuity для настройки доступов к объектам.

Подробнее о ролевой модели: Ролевая модель

Ключ (Key) - это бизнес-идентификатор (то есть текстовый ключ) объекта в системе.

Тэг (Tag) - это название версии объекта.

Бизнес тэг (Business tag) - это метка, которая может быть добавлена к любому объекту системы для использования при фильтрации.

Continuity SDK - python-библиотека предоставляющая полный функционал для удобной работы с API в интерфейсе Jupyter.

Сервисная задача (Service task) - объект типа task, который позволяет настраивать автоматическое исполнение задачи на базе python-скрипта.

Подробнее о настройке сервисных задач: Сервисные задачи

3.1.3 Граф Continuity

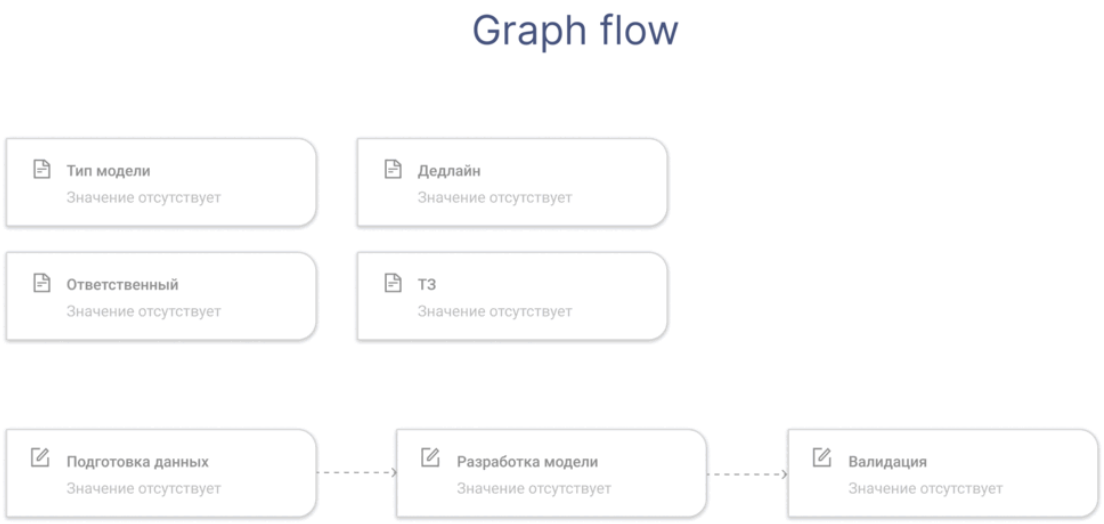
Workflow проекта – это ориентированный граф, состоящий из нод, которые содержат этапы, и ребер, которые задают направление процесса.

В широком смысле node (ноды) – это контейнеры объектов Continuity. Например, в рамках проекта они образуют граф из этапов.

Граф инициализируется при создании проекта пользователем в соответствии с шаблоном, после этого появляется первый этап. Новая нода создается после того, как завершилась предыдущая. Система проверяет возможность запуска новой ноды на основе матрицы достижимости графа и контекстов.

Условия переода на следующую ноду задается с помощью контекста Condition.

Подробнее о контекстах: Контексты



3.1.4 Ролевая модель

Ролевая модель Continuity - комбинация прав и ролей системы, описывающая доступ пользователей к тем или иным объектам.

Авторизация пользователей реализована на базе Keycloak.

ПРАВА ДОСТУПА

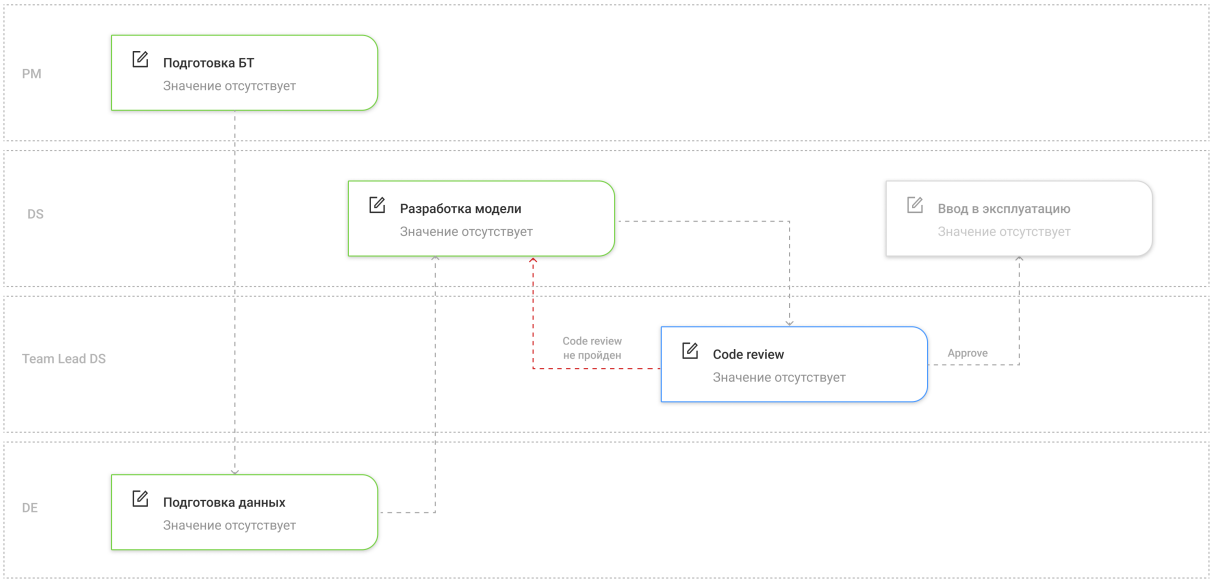
Owner имеет неограниченные права на объект. Создатель объекта обычно назначается владельцем по умолчанию.
Assigner имеет право право редактировать информацию, заполнять артефакты, завершать задачи.
Reader может видеть объект в каталоге, переходить на его страницу и смотреть информацию о нем.

Настройка доступа к объекту складывается из двух составляющих:
1. Выбор пользователя или группы прользователей (из списка, предложенного Keycloak)
2. Выбор прав (owner, assigner, reader)

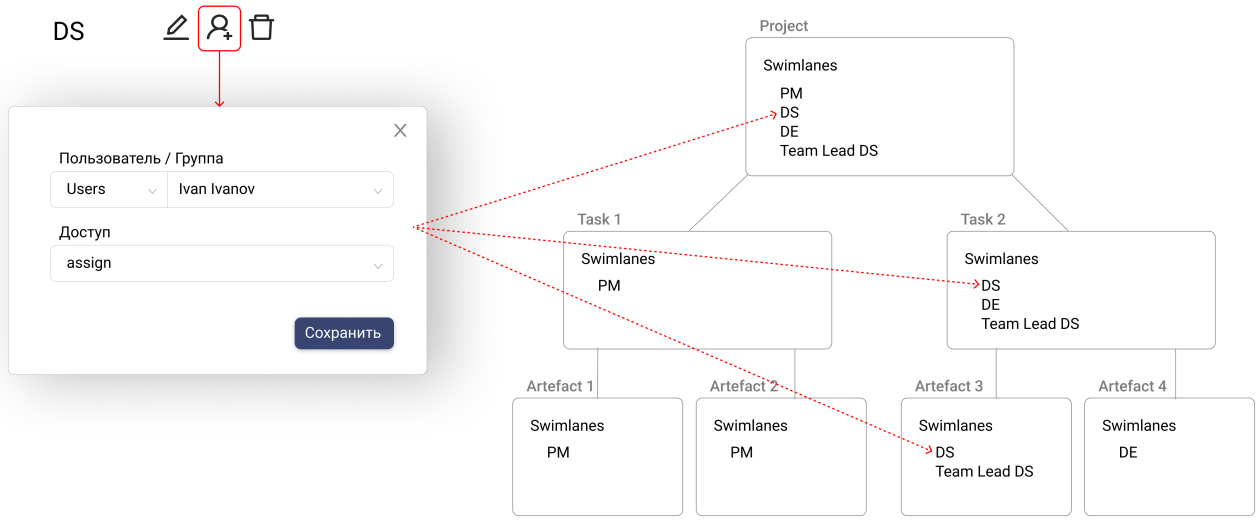
Для автоматического распространения настроенного доступа сразу на несколько объектов в рамках одного проекта существует система Swimlane.

SWIMLANE

Swimlane - это инструмент Continuity для настройки ролевого доступа к объектам.
Свимлейн дает возможность объединить объекты во множество для настройки доступа к ним из одной точки.



Таким образом заданные пользователем в свимлейне права доступа пробрасываются по всем объектам в рамках этого свимлейна.



Как настроить Swimlane: *Настройка доступов*

3.1.5 Continuity SDK

Continuity SDK - это Python фреймворк для работы с приложением из кода.

SDK дает возможность:

1. Получить список шаблонов в системе
2. Узнать количество шаблонов в системе
3. Получить информацию о конкретном шаблоне
4. Получить список объектов в системе с фильтрацией по типу
5. Узнать количество объектов в системе
6. Получить информацию о конкретном объекте
7. Получить список объектов, созданных по конкретному шаблону
8. Получить список всех дочерних объектов конкретного объекта или шаблона
9. Получить родителя любого уровня для конкретного объекта или шаблона
10. Создать проект из шаблона
10. Залогировать артефакт в систему
11. Завершить задачу
12. Настроить handler для запуска функциональной сервисной задачи

Для работы с фреймворком необходимо установить пакет klmg_continuity_sdk.

Service task handler

Система Continuity позволяет настраивать объекты, которые работают с помощью внешнего сервиса и выполняют функции.

Для того, чтобы при запуске runtime-объекта исполнялся какой-либо скрипт, необходимо настроить handler по ключу и версии шаблона объекта. Handler слушает запросы к системе и запускает скрипт при переводе объекта, созданного по этому шаблону, в статус in progress.

Руководство по работе с SDK: Ролевая модель

3.2 Шаблоны

3.2.1 Работа с шаблонами

kolmogorov.ai Continuity предоставляет возможность моделировать workflow через шаблоны для переиспользования объектов.

Туториал по моделированию шаблона: *Конструктор шаблонов*

В системе существует 3 типа объектов. Их можно шаблонизировать:

Объект	Название	Описание
Project	Проект	Объединяет объекты в рамках одной бизнес-задачи
Task	Задача	Описывает задачу как этап workflow
Artefact	Артефакт	Настраивает параметры артефакта, такие как тип, обязательность заполнения, дефолтное значение и т.п.

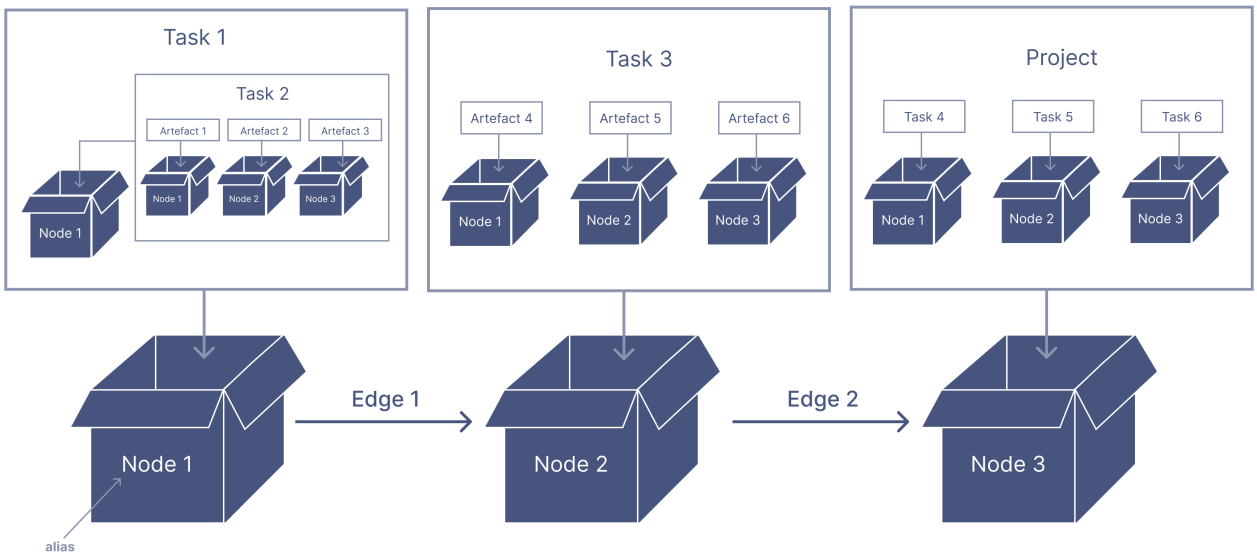
СОДЕРЖИМОЕ ШАБЛОНА

Шаблон состоит из элементов, называемых узлами (node), и имеет иерархическую структуру. Узлы являются контейнерами для объектов и могут быть вложены в другие объекты.

Так в проекте могут содержаться задачи, которые в свою очередь разбиваются на подзадачи, внутри которых могут быть вложены артефакты. При этом Continuity позволяет комбинировать объекты разных типов на одном уровне.

Например, пользователю необходимо создать шаблон проекта. У него есть ключ, версия, название, описание. Кроме этой мета информации проект может содержать настройки прав доступа, а также списки нод (node) и связей между ними (edge). В ноды добавляются объекты-задачи, которые также могут иметь свои ноды, и так далее по нисходящей иерархии.

Project template creation



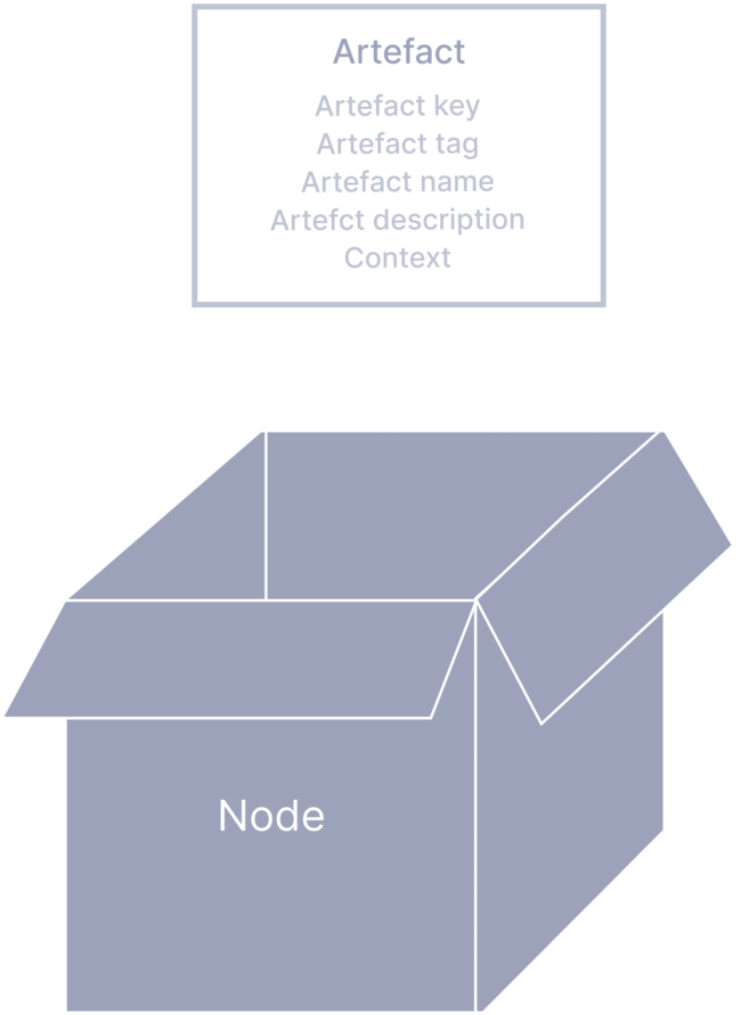
Каждая нода имеет идентификатор - **alias**, который является уникальным только в рамках родительского объекта. С его помощью можно переиспользовать один и тот же шаблон внутри одного объекта (например, повторение задачи в рамках проекта).

Contexts

У каждой сущности в системе может быть контекст. Он описывает особые свойства сущности. Например, такие как тип артефакта или тип задачи, условия перехода и т.п. Контексты прописываются в шаблоне объекта наряду с мета информацией.

Подробнее о контекстах: Контексты

СОЗДАНИЕ ШАБЛОНА ЗАДАЧИ



ВЕРСИИ ШАБЛОНА

3.2.2 Контексты

Контекст описывает свойство сущностей. Он задается в шаблоне наряду с мета информацией об объекте.

Туториал по моделированию шаблона: *Создание шаблона*

Типы контекстов Continuity

Контекст	Объект	Описание
type	artefact	Тип артефакта. Может принимать значения: string, dropdown, bool, link, date, file, user
type	task	Тип задачи (user/service)
require	artefact	Является ли артефакт обязательным (true/false)
default	artefact	Устанавливает дефолтное значение (строка)
value	artefact	Относится только к артефакту типа dropdown: добавляет значение в выпадающий список (строка)
multi	artefact	Относится только к артефакту типа dropdown: дает возможность выбора нескольких значений списка (true/false)
condition	edge	Добавление условия перехода на следующий объект (выражение вида: '{artefact_name}'=='value')

3.3 Объекты

3.3.1 Runtime

Runtime-объекты Continuity - это экземпляры сущностей, созданные со своими уникальными ключом и версией.

Данные сущности могут иметь несколько статуов:
1. Active - in progress
2. Completed - done
3. Archived - удаленный с возможностью восстановления.

Объекты в Cntinuity могут быть созданы из *шаблона*.

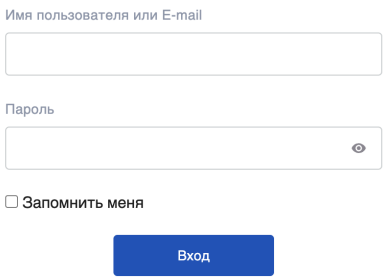
Подробнее о том, как запустить проект: *Создание проекта*

3.4 Работа с приложением

3.4.1 Пользовательский интерфейс

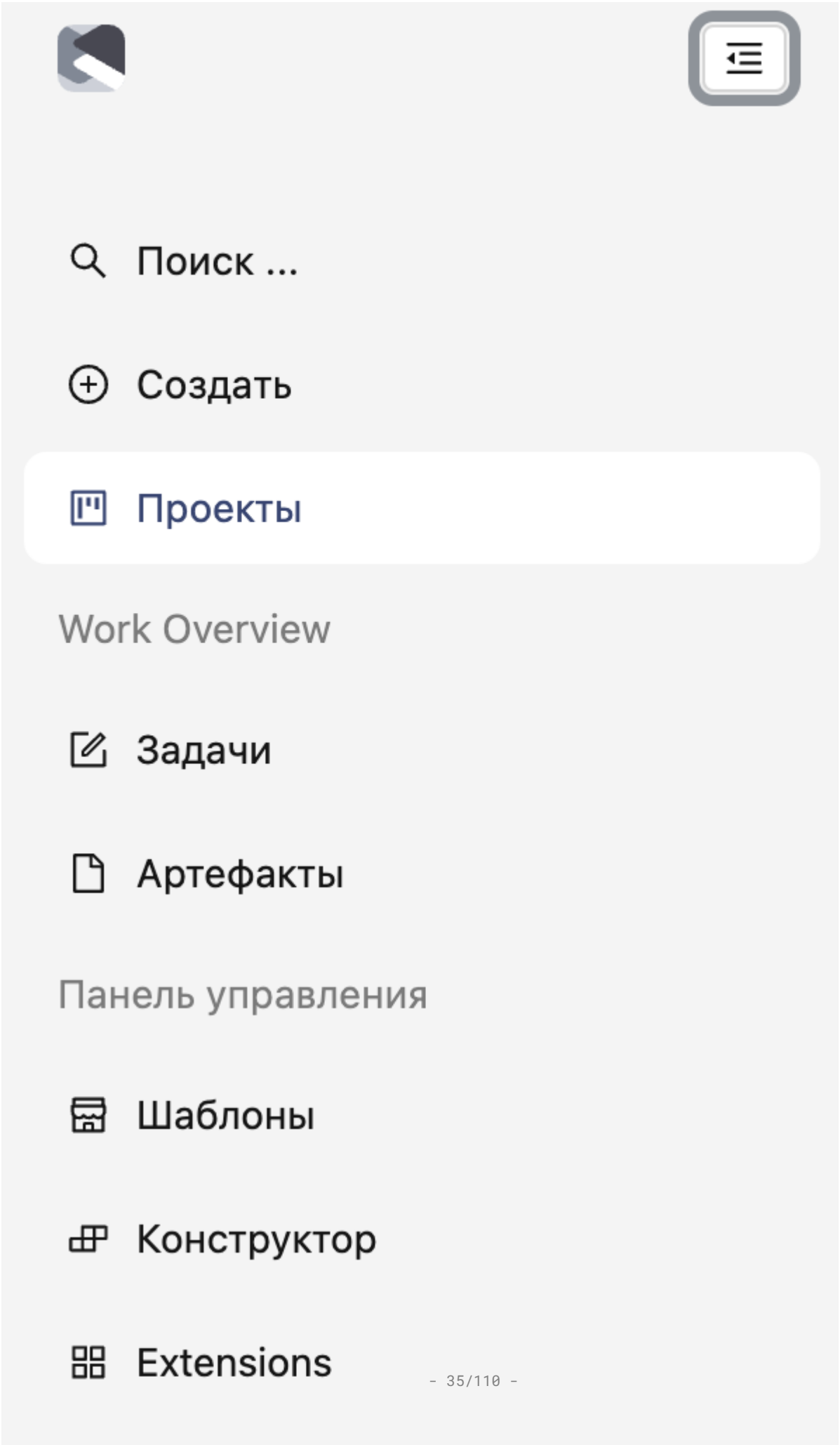
Авторизация

Чтобы войти в систему, введите логин и пароль, который соответствует вашей учетной записи в Keycloak.



Меню

Меню Continuty находится в левой части экрана. По умолчанию меню свернуто и представляет собой набор значков, при наведении на каждый из которых вы увидите название соответствующего раздела. Чтобы развернуть панель, нажмите на значок в левом нижнем углу экрана.

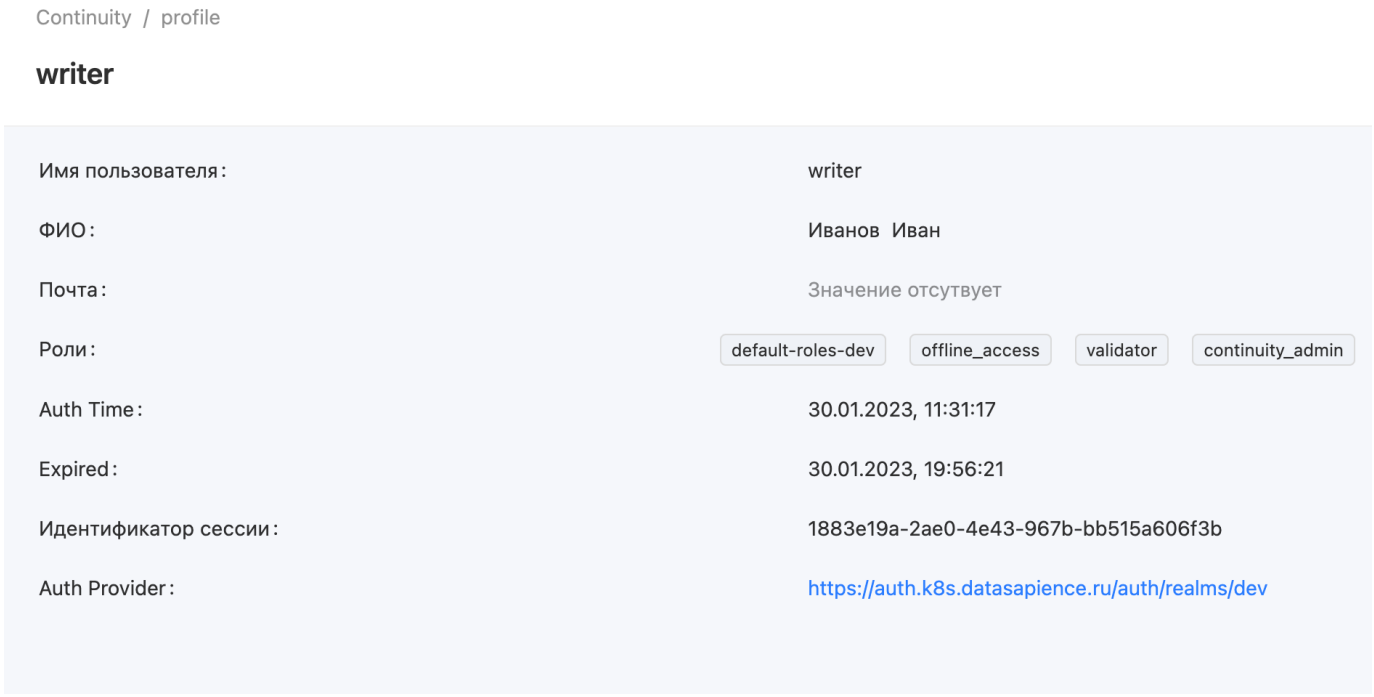

Поиск ...
Создать
Проекты
Work Overview
Задачи
Артефакты
Панель управления
Шаблоны
Конструктор
Extensions

Тема Continuity

В Continuity доступны две темы: светлая и темная. Чтобы переключить тему, нажмите на соответствующую кнопку в нижней части меню.

Профиль пользователя

При нажатии на значок учетной записи вы перейдете в свой профиль. Тут содержатся ваши учетные данные, роли в системе, информация о сессии.



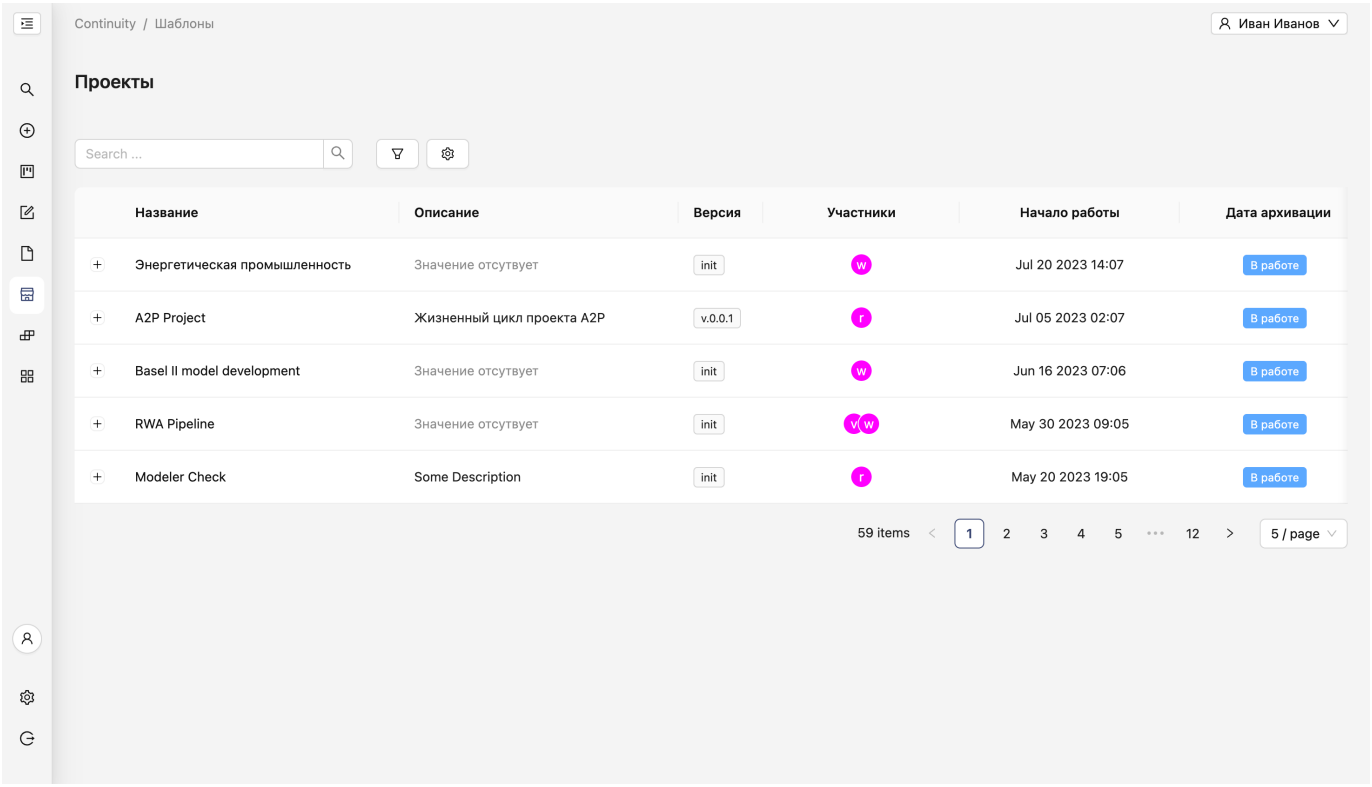
Каталоги

Чтобы перейти в каталог проектов/задач/артефактов выберете соответствующую кнопку в меню. Каталог представляет собой список объектов с метаинформацией о них. Все каталоги построены аналогичным образом. Это таблица с возможностью переключаться между страницами - по умолчанию на странице отображено 5 объектов. Изменить это можно, выбрав соответствующее значение в выпадающем списке пагинатора.

Над таблицей находятся настройки каталога, с помощью которых вы можете установить периодичность обновления страницы, отфильтровать объекты по тегам, вывести только свои проекты.
Каталоги хранят историю изменения каждого объекта, посмотреть ее можно при нажатии на соответствующий значок в левой части таблицы.

Каталог проектов

В таблице с проектами содержится информация о названии, описании, ключе, версии, статусе и владельце каждого проекта.
Владелец - это пользователь, который создал проект.
Статус проекта отображает этап, на котором находится проект.

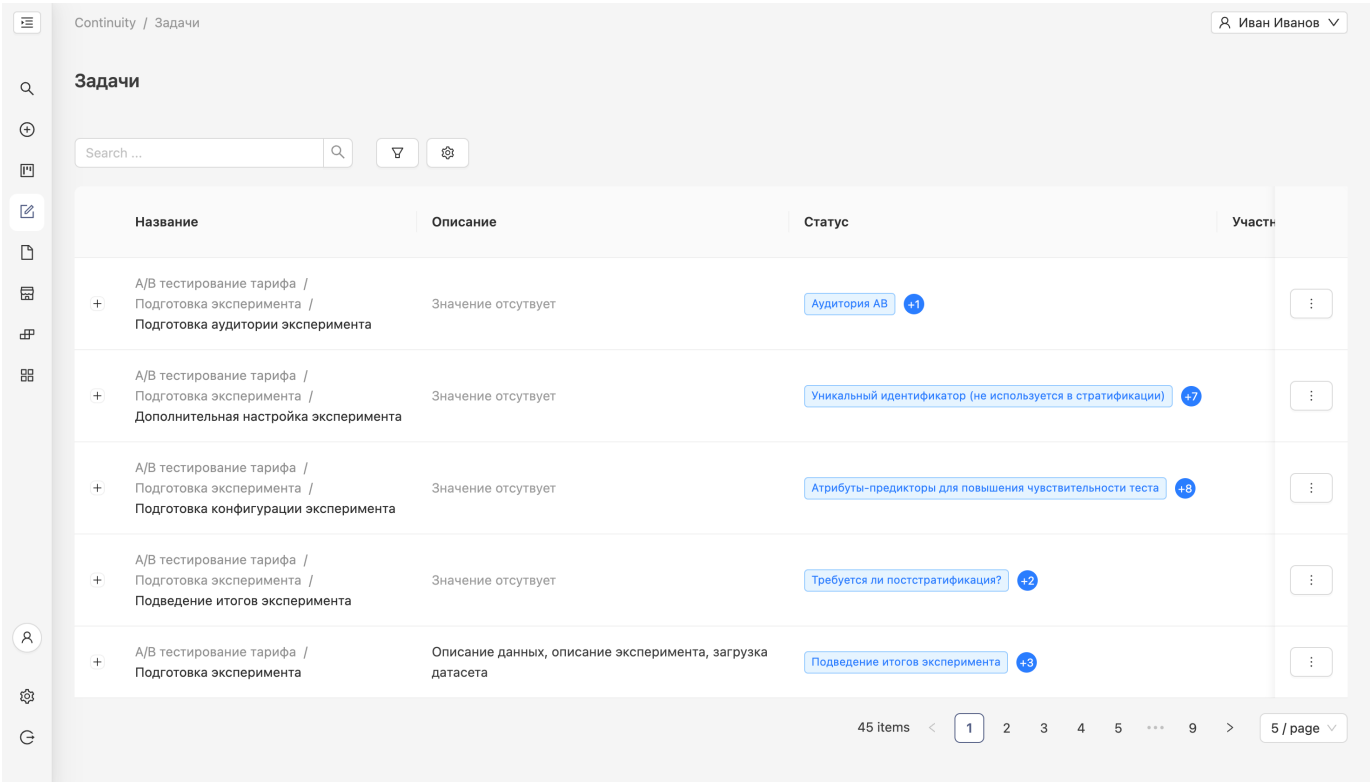


Каталог задач

В таблице с задачами содержится информация о названии, описании, ключе, версии, статусе и исполнителе каждой задачи.

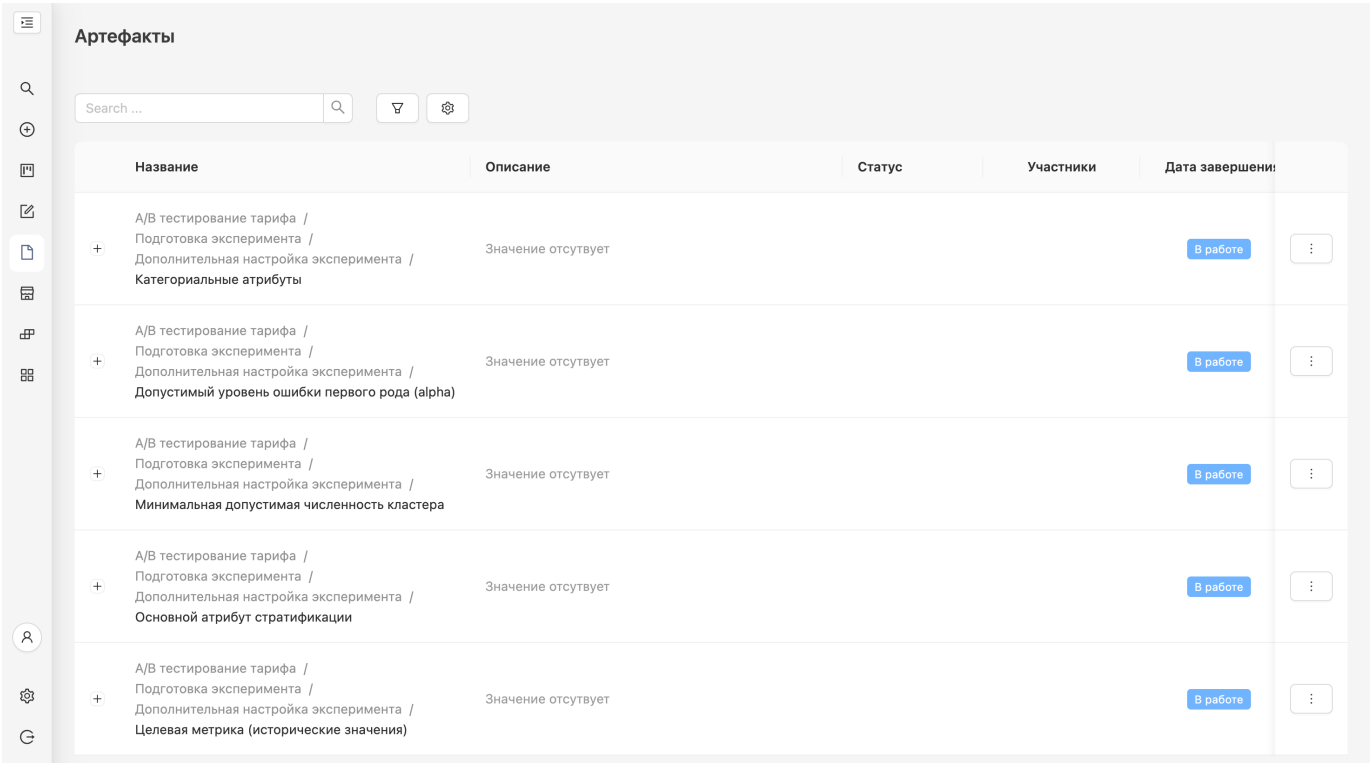
В **статусе** задачи отображается состояние задачи - задача может быть в процессе или завершена.

Исполнителем является назначенный на задачу пользователь.



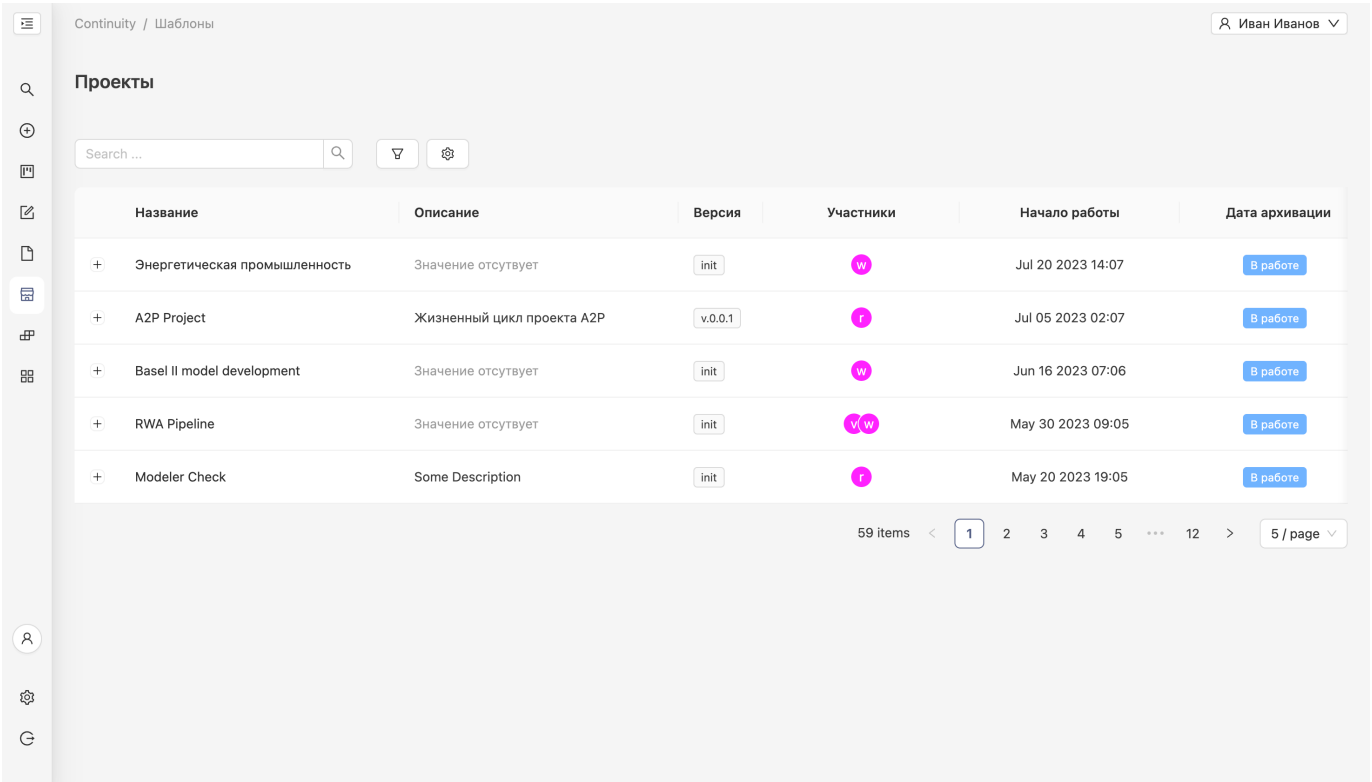
Каталог артефактов

В таблице с артефактами содержится название, значении артефакта и о проекте, за которым закреплен объект. Значение артефакта можно скопировать, нажав на значок опций в правой части таблицы. Там же отображается возможность скачивания артефакта типа файл.



Каталог шаблонов

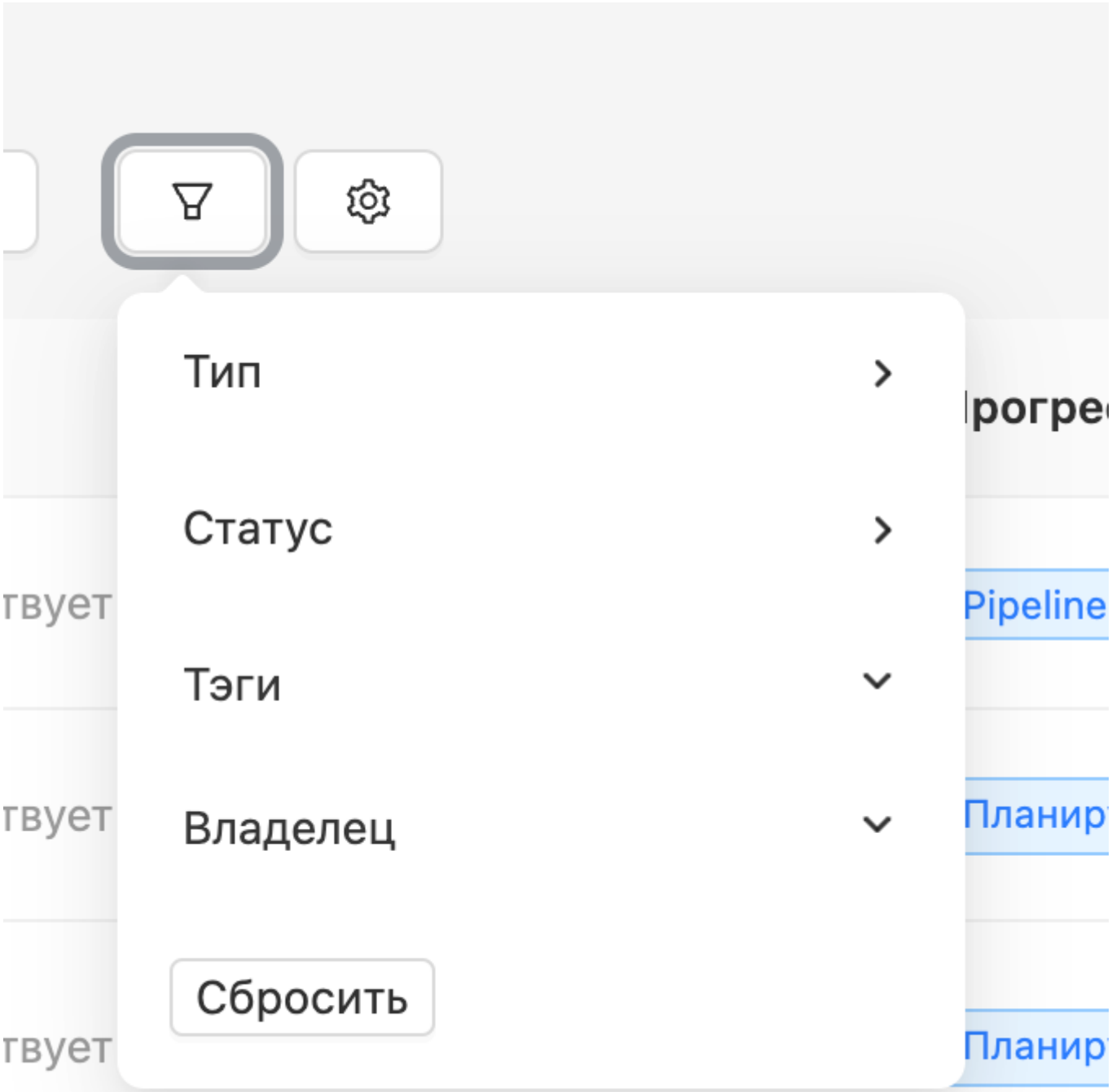
В таблицах с шаблонами объектов содержится название, описание, ключ, версия и дата создания шаблона.



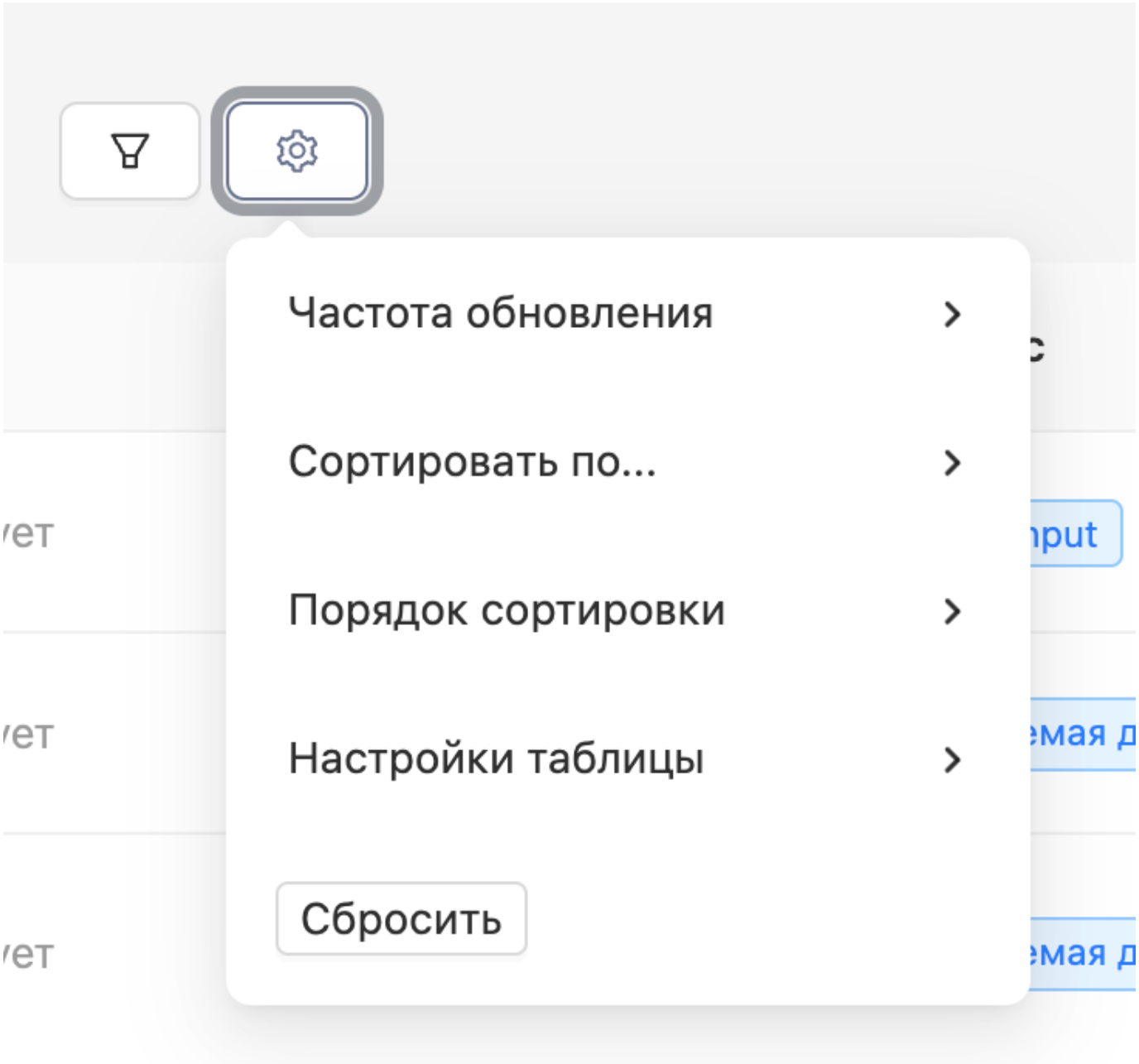
Фильтры и параметры поиска

Для более удобного поиска объектов в каталоге пользователю предлагается набор настроек поиска и фильтрации.
1. Чтобы открыть фильры, нажмите на значек фильтра.
2. Чтобы открыть параметры поиска, нажмите на значок шестеренки.

Вы можете отфильтровать объекты в каталоге по:
1. Тегу
2. Типу объекта
3. Владельцу
4. Статусам



В параметрах поиска представлена возможность настройки:
1. Частоты обновления объектов в каталоге
2. Полей для сортировки
3. Типа сортировки
4. Вида таблицы

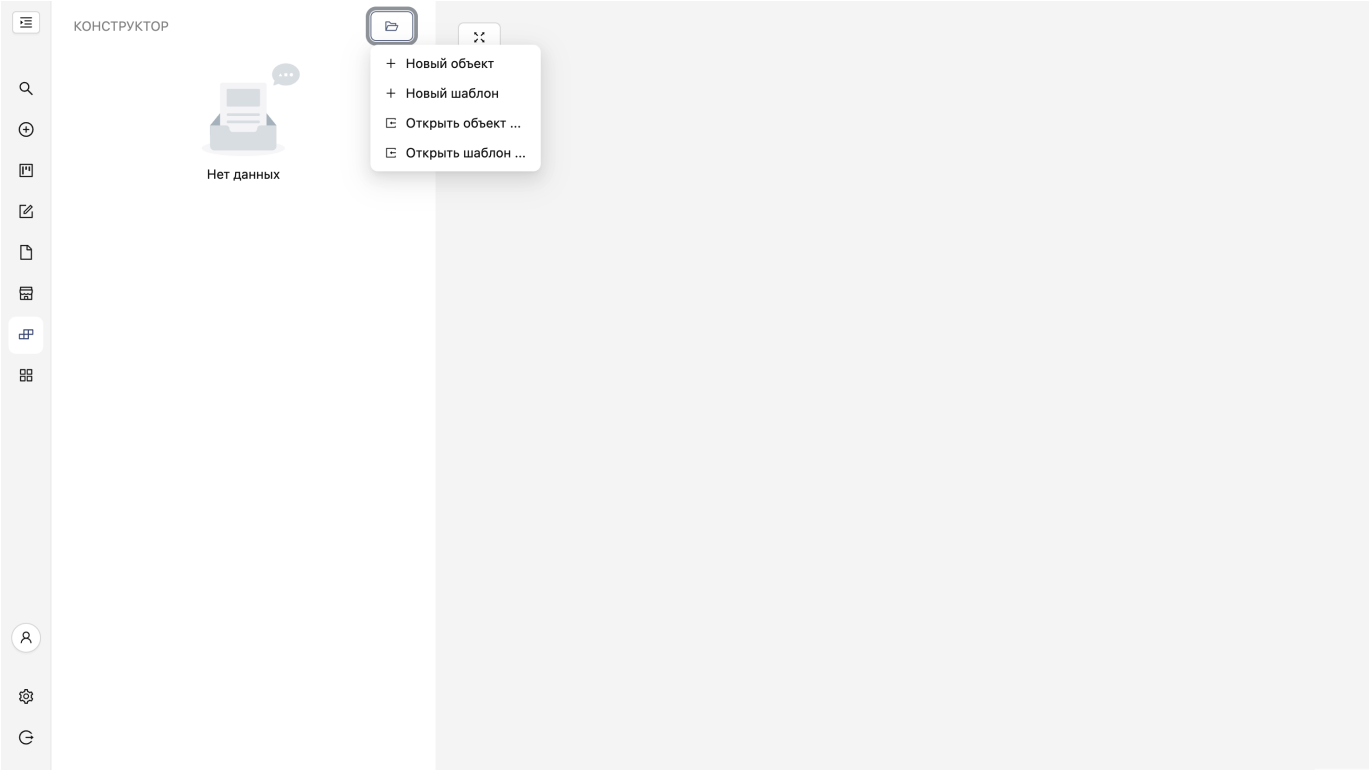


Конструктор

Интефейс предполагает возможность редактирования шаблонов и объектов пользователем через Конструктор.

Перейдите в Конструктор, нажав на соответствующую кнопку на левом меню. Вы увидите панель управления, где настраиваются мета параметры объекта, и рабочее поле для работы с нодами и ребрами workflow.

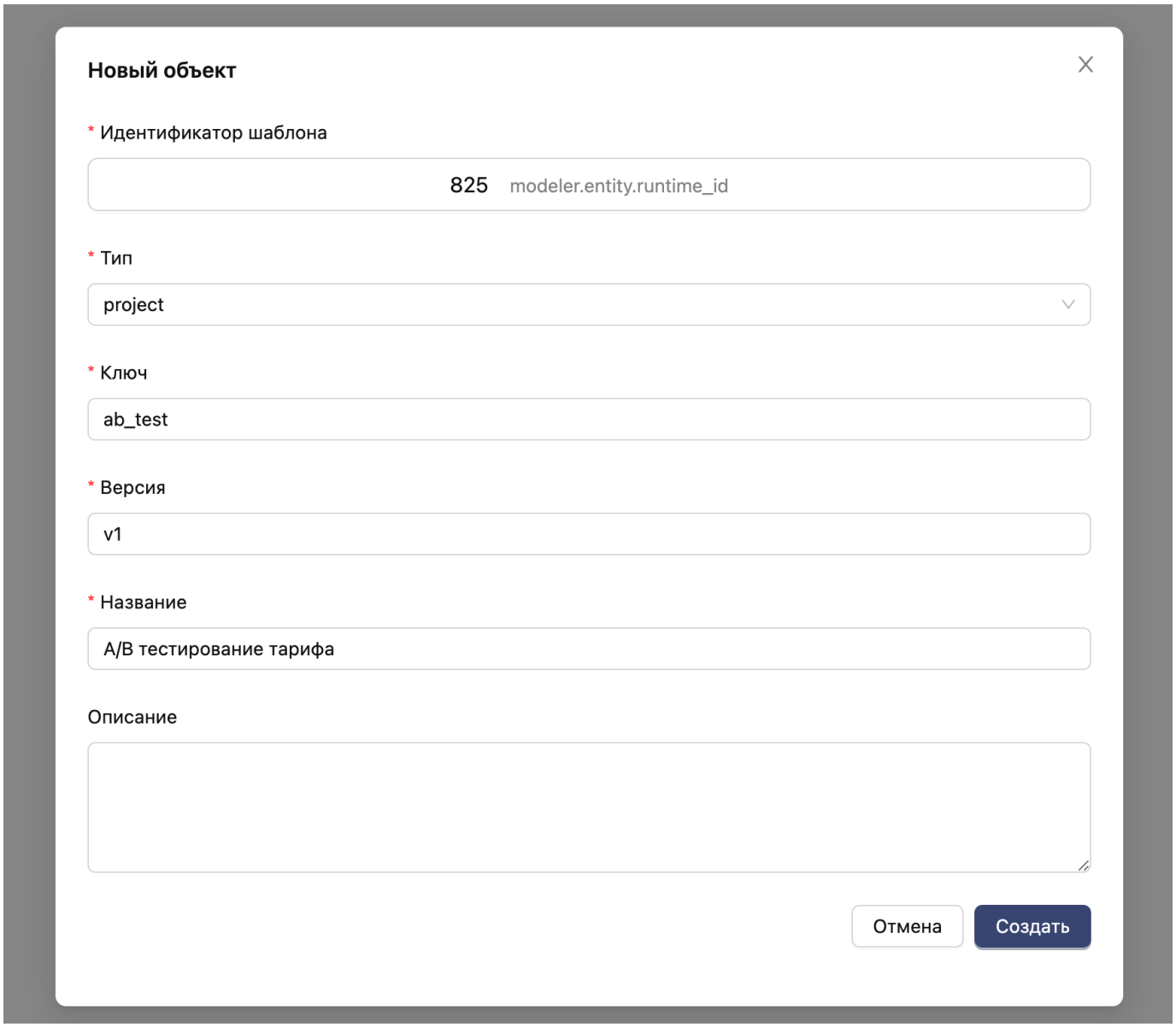
Для того, чтобы отредактировать или создать новый объект/шаблон, выберете соответствующую опцию в выпадающем меню в правом верхнем углу панели инструментов.



Подробнее:
Создание и редактирование шаблона

Запуск runtime объекта

В системе доступен бытрый запуск нового runtime-объект из шаблона. Для того, чтобы это сделать, нажмите кнопку "Создать" в правом меню.

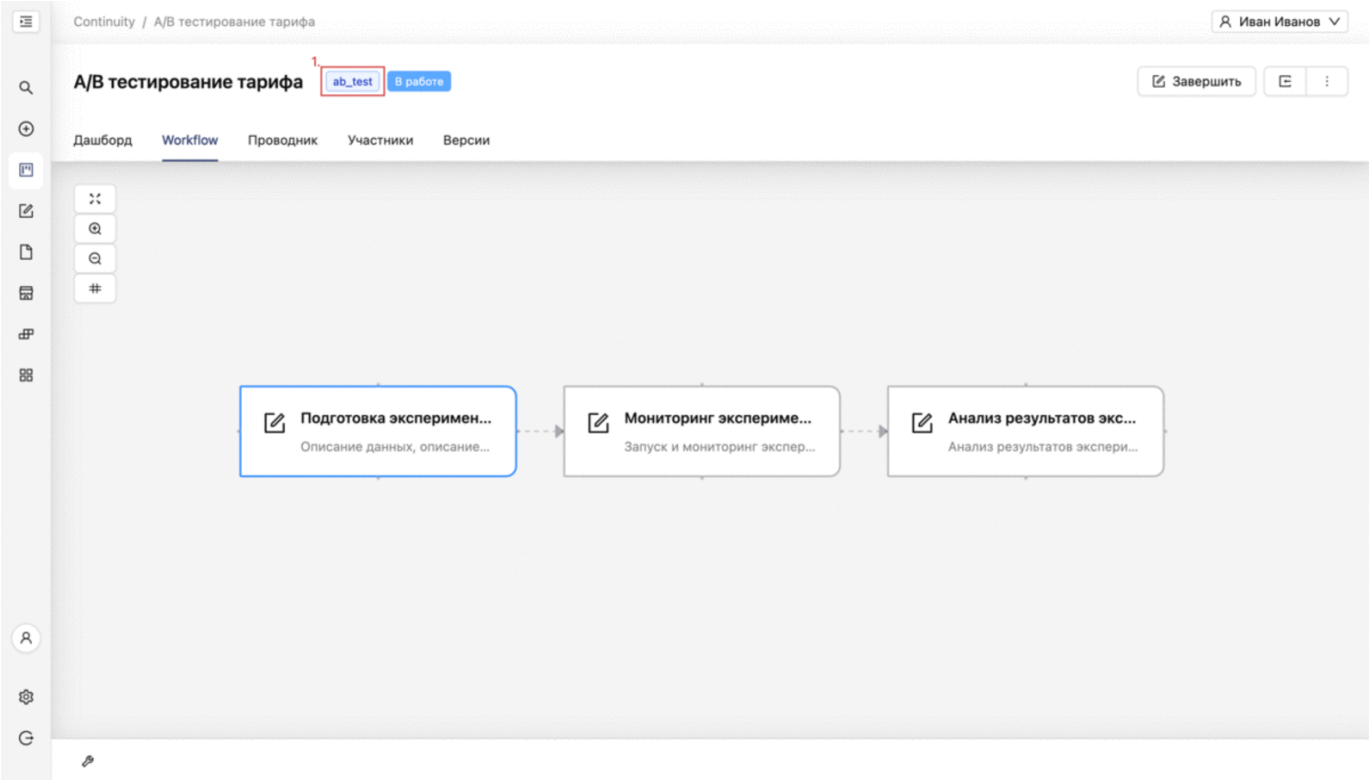


В появившемся модальном окне выберете шаблон и введите мета-информацию.

Подробнее: Запуск проекта

Обзор страницы объекта

Страницы всех объектов (Проект, Задача, Артефакт) устроены алналогичным образом.
Ниже описаны основные элементы страницы.



1. Теги, добавленные пользователем
2. Статус объекта
3. Раздел "Дашборд": на данной вкладке содержится статистическая информация об объекте
4. Workflow-граф, на котором отображены все дочерние объекты, со связями между ними и с возможностью перехода к каждому из них
5. Раздел "Проводник" представляет собой каталог вложенных объектов (например, задач и артефактов у проекта)
6. В разделе "Участники" производится настройка доступов пользователей к объекту
7. В разделе "Версии" содержится каталог версий объекта
8. Панель инструментов: дает возможность центрировать, увеличивать и уменьшать граф, а также добавлять сетку
9. Кнопка "Завершить" позволяет перевести объект в статус done
10. Кнопка "Скачать манифест" позволяет скачать JSON-представление объекта
11. Дополнительные опции (обновить страницу, синхронизировать граф, архивировать объект)
12. Кнопка открытия панели дополнительных инструментов: дает возможность увидеть все артефакты объекта, посмотреть/оставить комментарии к объекту, посмотреть список свойств объекта, посмотреть JSON-представление объекта

Окончание сессии

Для выхода из системы нажмите на соответствующую кнопку в нижней части меню.

3.4.2 Создание и редактирование шаблонов

В Continuity существуют 3 типа объектов: проект, задача, артефакт. У каждого объекта есть определенный набор свойств, регламентирующий его работу.

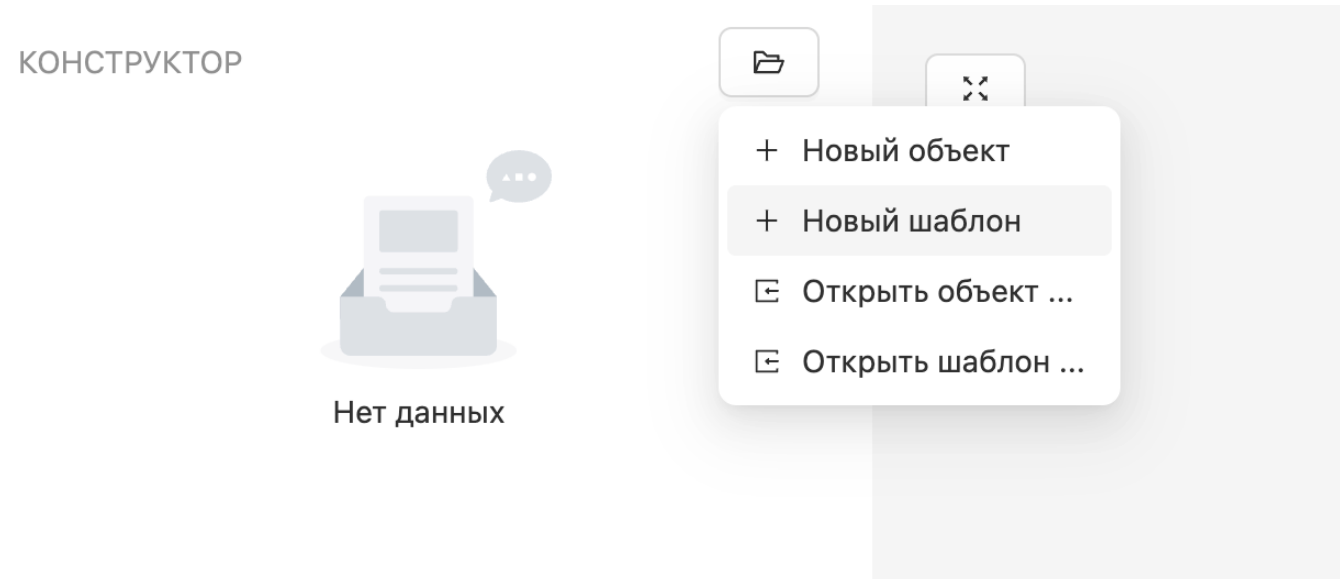
Например, при взаимодействии пользователя с интерфейсом workflow ему приходится заполнять различные формы для сохранения важных артефактов разработки. Эти артефакты могут иметь вид текста, ссылки, файла, даты, упоминания пользователя, бинарного значения. Тип информации вводимой пользователем - это свойство артефакта, определяющее способ ввода и отображение в каталоге.

В Continuity у всех объектов есть свой набор возможных свойств. Таким же образом настраиваются ноды и ребра - они помогают организовать и автоматизировать workflow.

Для сохранения свойств объекта в целях его переиспользования существует возможность добавления шаблонов, которые можно настроить через Конструктор Continuity.

НОВЫЙ ШАБЛОН

Чтобы создать новый шаблон перейдите в конструктор, выберете опцию "Новый шаблон" в правом верхнем углу панели управления.

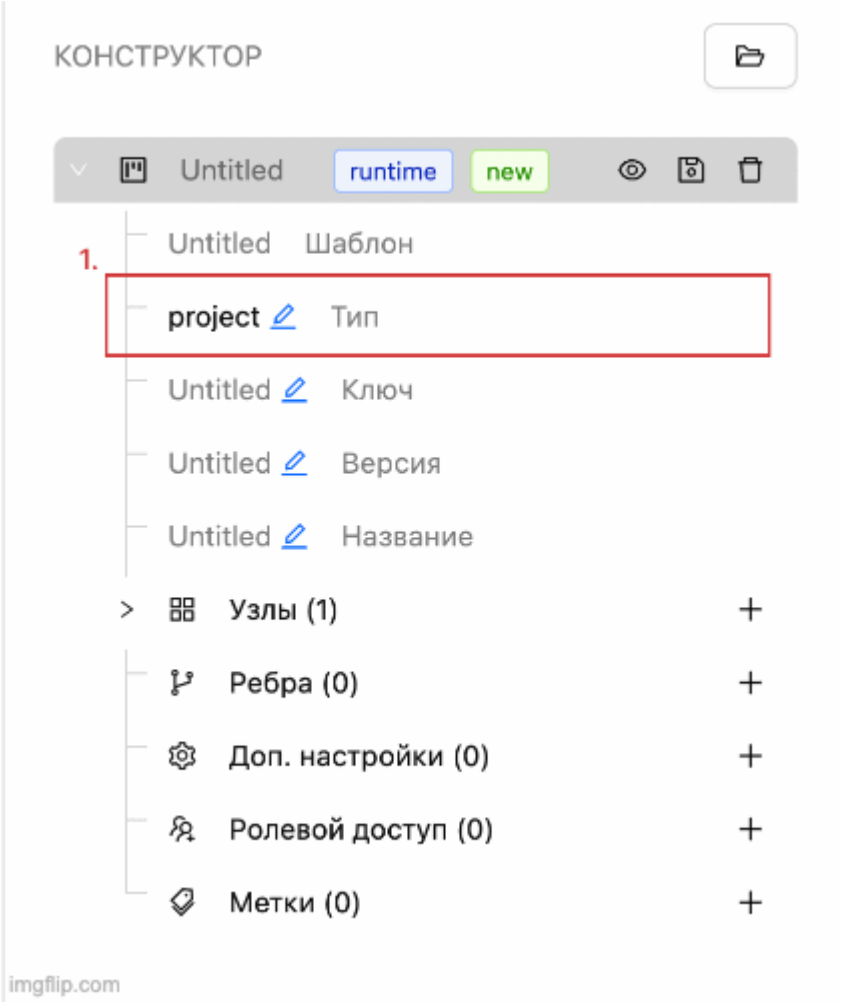


В конструкторе можно работать сразу над несколькими объектами, в том числе и связанными между собой как родитель-ребенок.

ОБЗОР РЕДАКТОРА ОБЪЕКТА

В редакторе объекта есть 7 основных разделов:

1. Выбор типа объекта (проект, задача, артефакт)
2. Заполнение/редактирование основной информации
3. Добавление/удаление/редактирование узлов
4. Добавление/удаление/редактирование ребер
5. Дополнительные настройки
6. Добавление/удаление/редактирование ролей
7. Добавление/удаление/редактирование меток



ВЫБОР ТИПА ОБЪЕКТА

1. Разверните редактор объекта, кликнув на его название (по умолчанию "Untitled")
2. В поле "Тип объекта" выберете из выпадающего меню необходимое значение (project/task/artefact)

РЕДАКТИРОВАНИЯ ИНФОРМАЦИИ ОБ ОБЪЕКТЕ

1. Разверните редактор шаблона, кликнув на его название (по умолчанию "Untitled")
2. Разверните вкладку "Основная информация"
3. Введите ключ шаблона, кликнув на соответствующее поле (Ключ - это бизнес-идентификатор. Может состоять из латинских букв, цифр, тире и нижних подчеркиваний. Для сохранения нажмите Enter
4. При необходимости поменяйте версию объекта (по умолчанию init). Для сохранения нажмите Enter
5. Введите название. Для сохранения нажмите Enter

ДОБАВЛЕНИЕ ДОЧЕРНИХ ОБЪЕКТОВ

1. Разверните редактор объекта, кликнув на его название (по умолчанию "Untitled")
2. Нажмите на кнопку плюса напротив вкладки "Узлы", вы увидите, что значение на счетчике узлов увеличилось
3. Раскройте вкладку "Узлы", кликнув на нее
4. Раскройте вкладку необходимого узла, кликнув на него
5. Если объект уже имеется в системе, нажмите на кнопку "Выбрать из каталога" - выберете из каталога необходимый объект

6. Если нужного объекта нет в системе, нажмите на кнопку новый объект. После этого под меню редактирования основного объекта появится меню редактирования нового дочернего объекта. Отредактируйте новый объект и сохраните его (аналогично основному объекту)

ДОБАВЛЕНИЯ СВЯЗИ МЕЖДУ УЗЛАМИ (WORKFLOW)

1. Разверните редактор объекта, кликнув на его название (по умолчанию "Untitled")
2. Нажмите на значок глаза напротив названия создаваемого/редактируемого объекта - на рабочем поле справа отобразятся все добавленные в объект узлы
3. Наведите курсор на точку сбоку стартового объекта, нажмите на нее и протяните до целевого объекта - вы увидите, что появилась стрелка, задающая направление workflow
4. Проделайте это со всеми объектами, которые вы хотите объединить в workflow
5. Нажмите кнопку "Сохранить" в левом верхнем углу рабочего поля - вы увидите, что на панели редактирования во вкладке "Ребра" появились связи между узлами

Если в один узел входят два и более ребер параллельно: 1. Разверните вкладку Ребра 2. Найдите все параллельные ребра, входящие в узел 3. Для каждого ребра нажмите кнопку плюса напротив раздела Дополнительные настройки 4. В появившемся поле Group проставьте значение **true**

ДОБАВЛЕНИЯ ЦИКЛА (WORKFLOW)

Примечание: функция для отката процесса на более ранний этап 1. Разверните редактор объекта, кликнув на его название (по умолчанию "Untitled") 2. Разверните вкладку Ребра 3. Найдите ребро, возвращающее процесс на более ранний этап 4. Кликните на флаг "is_rollback" - вы увидите, что данный флаг стал активным 5. При необходимости добавьте условие на узел отката (см. Добавление условия перехода по workflow)

ДОПОЛНИТЕЛЬНАЯ НАСТРОЙКА ОБЪЕКТОВ (ОПЦИИ НАСТРОЕК СМ. НИЖЕ)

1. Разверните редактор объекта, кликнув на его название (по умолчанию "Untitled")
2. Нажмите на кнопку плюса напротив вкладки "Доп. настройки", вы увидите, что значение на счетчике настроек увеличилось
3. Раскройте вкладку "Доп. настройки", кликнув на нее - вы увидите список контекстов (что такое контексты - см. ниже)
4. Раскройте контекст, кликнув на него
5. Введите название контекста (см. ниже список названий контекстов)
6. Введите значение контекста (см. ниже возможные значения контекстов)
7. Введите описание контекста

НАСТРОЙКА РОЛЕЙ И ДОСТУПОВ

1. Разверните редактор объекта, кликнув на его название (по умолчанию "Untitled")
2. Нажмите на кнопку плюса напротив вкладки "Участники" - вы увидите, что значение на счетчике участников увеличилось
3. В поле "Swimlane" введитне название swimlane (что такое swimlane - см. ниже)
4. При необходимости в поле "Swimlane Group" проставьте пользовательскую группу Kaycloak - тогда доступ к объекту будут иметь пользователи только из этой группы
5. При необходимости вы можете сразу добавить пользователей и права доступа на swimlane: для этого нажмите на кнопку плюса напротив поля "Пользователи", затем, кликнув на появившуюся форму, введите пользователя/группу из Keycloak и уровень доступа (см. ниже список прав доступа)

ДОБАВЛЕНИЕ УСЛОВИЯ ПЕРЕХОДА ПО WORKFLOW

1. Разверните редактор объекта, кликнув на его название (по умолчанию "Untitled")
2. Разверните вкладку Узлы
3. Разверните необходимый узел (условие необходимо ставить узел, который необходимо запустить по условию)

4. С помощью кнопки плюса напротив раздела Дополнительные настройки добавьте новый контекст
5. В названии укажите слово "condition" (см. ниже настройки и контексты)
6. В значении контекста введите следующее выражение: "**{artefact_key}" == "artefact_value**", где artefact_key - код артефакта, по которому будет работать условие, а artefact_value - это значение, которое должен принять этот артефакт в workflow, чтобы условие выполнилось. **Примечание: код артефакта и возможные значения (при условии, что тип артефакта – dropdown) можно посмотреть, открыв шаблон артефакта в конструкторе.**

МЕТКИ (ТЭГИ ДЛЯ ФИЛЬТРАЦИИ В СИСТЕМЕ)

1. Разверните редактор объекта, кликнув на его название (по умолчанию "Untitled")
2. Нажмите на кнопку плюса напротив вкладки "Метки" - вы увидите, что значение на счетчике меток увеличилось
3. Раскройте вкладку "Метки" - введите метку в появившемся поле

СОХРАНЕНИЕ ОБЪЕКТА

1. Нажмите на значок сохранения напротив названия создаваемого/редактируемого объекта
2. В появившемся окне вы можете посмотреть получившийся json-объект, затем нажмите кнопку "Сохранить"
3. Перейдите к созданному/отредактированному объекту, нажав на кнопку "Перейти"

УДАЛЕНИЕ

Для удаления объекта/узла/ребра/контекста нажмите на значок урны.

РЕДАКТИРОВАНИЕ РАСПОЛОЖЕНИЯ ОБЪЕКТОВ НА WORKFLOW

1. Разверните редактор объекта, кликнув на его название (по умолчанию "Untitled")
2. Нажмите на значок глаза напротив названия создаваемого/редактируемого объекта – на рабочем поле справа отобразятся все добавленные в объект узлы
3. Перемещайте объекты по полю, нажав и удерживая их левой кнопкой мыши

Настройка свойств объектов

Контекст объекта (Context) - свойство объекта. С помощью контекстов можно настраивать объекты: например, устанавливать тип задачи, тип артефакта и т.п.

Типы свойств Continuity

Контекст (Название)	**Объект**	**Комментарий**
type	artefact	Тип артефакта. Может принимать значения: string, dropdown, bool, link, date, file, user
type	task	Тип задачи (user/service)
require	artefact	Является ли артефакт обязательным (true/false)
default	artefact	Относится только к артефакту типа dropdown: устанавливает дефолтное значение (строка)
value	artefact	Относится только к артефакту типа dropdown: добавляет значение в выпадающий список (строка)
multi	artefact	Относится только к артефакту типа dropdown: дает возможность выбора нескольких значений списка (true/false)
condition	edge	Добавление условия перехода на следующий объект (выражение вида: '{artefact_name}'=='value')

Swimlane

Swimlane - это инструмент Continuity для настройки доступов к объектам. Swimlane в объектах наследуется из шаблона или проставляется в конструкторе при создании объекта с нуля. В каждый swimlane можно назначить пользователей из корпоративного Keycloak на каждую группу доступа:

- read: дает возможность видеть объект в каталоге, переходить на его страницу и смотреть информацию о нем
- assign: дает право редактировать информацию, заполнять артефакты, завершать задачи
- owner: дает неограниченные права на работу с объектом

3.4.3 Запуск проекта по шаблону

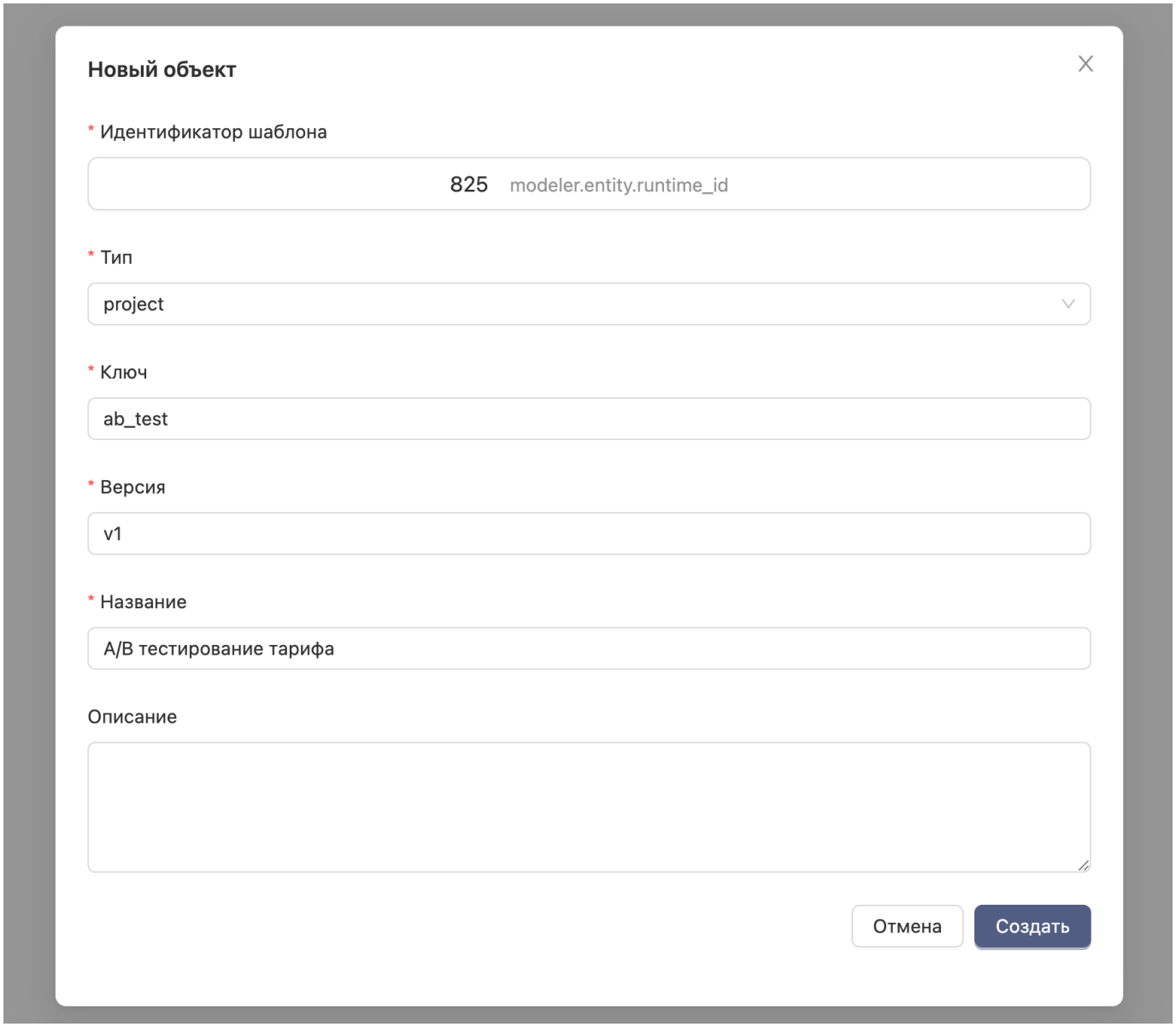
Для начала работы в Continuity необходимо создать проект. Для этого выполните шаги, описанные ниже.

ОТКРЫТИЕ ФОРМЫ СОЗДАНИЯ ПРОЕКТА

Выберите функцию “Создать” в основном меню приложения.

ВВЕДЕНИЕ ОСНОВНОЙ ИНФОРМАЦИИ О ПРОЕКТЕ

В форме создания проекта выберете необходимый шаблон из каталога, затем введите ключ, версию, название и описание.
Key (ключ) - это бизенес-идентификатор объекта. Допустимые значения: цифры, латинские буквы, символы пробела и нижнего подчеркивания.
Tag (тэг) - это версия объекта Допустимые значения: цифры, латинские буквы.
Название - произвольное название проекта.
Описание (опционально) - более подробное раскрытие назначения проекта.



3.4.4 Continuity SDK

Установка и подключение

Необходимо поставить в окружение разработки пакет Continuity SDK и настроить подключение:

```
from klmg_continuity_sdk import Session

session = Session(
    host='https://continuity-demo.k8s.datasapience.ru/api',
    username='rost',
    password='rost'
)
session.health()
```

Создание шаблона

```
from datetime import datetime
from klmg_continuity_sdk.schemas import TEMPLATE

template = TEMPLATE(
    ENTITY_TYPE='template_type',
    DIM_ENTITY_KEY='template_key',
    DIM_ENTITY_TAG=datetime.now().strftime('%a, %d %b %Y %H:%M:%S'),
    DIM_ENTITY_NAME='Template name'
)

session.create(template)
```

Также можно добавить следующие параметры: 1. DIM_ENTITY_DESC (описание шаблона)
2. IS_MAIN_VERSION (true/false)
3. NODES (список узлов в JSON представлении)
4. EDGES (список связей между узлами в JSON редставлении)
5. CONTEXT (дополнительные настройки в JSON представлении)

Пример JSON
Дополнительные настройки

Получение всех шаблонов системы

```
all = session.template.all(entity_type='project')
all
```

Возможные фильтры: 1. entity_type ('project', 'task', 'artefact') 2. owner (user_name) 3. status ('all', 'active', 'completed', 'archived')

Получение количества шаблонов

```
count = session.template.count(entity_type='project')
count
```

Возможные фильтры: 1. entity_type ('project', 'task', 'artefact') 2. owner (user_name) 3. status ('all', 'active', 'completed', 'archived')

Получение шаблона по ключу и версии

```
one = session.template.one(
    key=template.DIM_ENTITY_KEY,
    tag=template.DIM_ENTITY_TAG
)
one
```

Получение всех объектов, созданных по определенному шаблону

```
successors = session.template.successors(
    key=template.DIM_ENTITY_KEY,
```

```
    tag=template.DIM_ENTITY_TAG,
    entity_type=template.ENTITY_TYPE
)
successors
```

Создание RUNTIME объекта по шаблону

```
from datetime import datetime
from klmg_continuity_sdk.schemas import RUNTIME

runtime = RUNTIME(
    DIM_ENTITY_ID=template.DIM_ENTITY_ID,
    ENTITY_TYPE='entity_type',
    FCT_ENTITY_KEY='entity_key',
    FCT_ENTITY_TAG=datetime.now().strftime('%a, %d %b %Y %H:%M:%S'),
    FCT_ENTITY_NAME='Entity name'
)

session.create(runtime)
runtime
```

Получение количества RUNTIME объектов

```
count = session.runtime.count(entity_type=runtime.ENTITY_TYPE)
count
```

Получение всех RUNTIME объектов

```
all = session.runtime.all(entity_type=runtime.ENTITY_TYPE, limit=1)
all
```

Получение RUNTIME объекта по ключу и версии

```
one = session.runtime.one(
    key=runtime.FCT_ENTITY_KEY,
    tag=runtime.FCT_ENTITY_TAG
)
one
```

Получение всех дочерних объектов RUNTIME объекта

```
children = session.runtime.children(
    key=runtime.FCT_ENTITY_KEY,
    tag=runtime.FCT_ENTITY_TAG
)
children
```

Получение всех вышестоящих родителей RUNTIME объекта

```
parents = session.runtime.parents(
    key=runtime.FCT_ENTITY_KEY,
    tag=runtime.FCT_ENTITY_TAG
)
parents
```

Сохранение артефакта

```
artefact_template_key.ENTITY.FCT_ENTITY_DESC = "artefact_value"
artefact_template_key.ENTITY.IS_COMPLETE = True

session.update(task, is_complete=True)
```

Перевод сущности в статус done

```
session.update(task, is_complete=True)
```

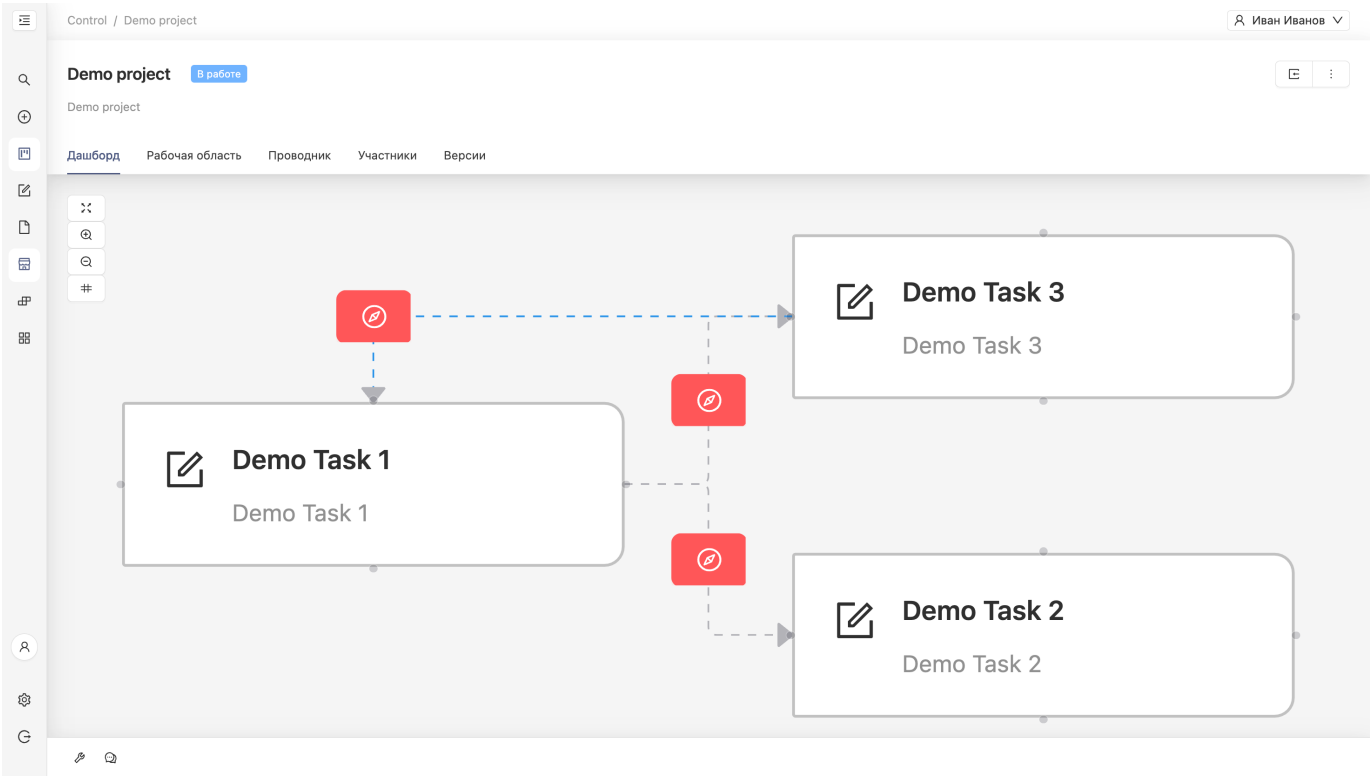
Пример настройки handler для сервисной задачи

```
@manager.subscribe(
    key=settings.TEMPLATE_ENTITY_KEY,
    tag=settings.TEMPLATE_ENTITY_TAG,
    entity_type='entity_type',
    interval=60
)
def function_name(entity: RUNTIME):

    artefact: RUNTIME_NODE = None

    for node in entity.NODES:
        if node.ALIAS == 'artefact_template_key':
            artefact = node

    if not artefact:
        return

    # Take all artefact successors keys of project
    args = session.runtime.inputs(
        key=entity.FCT_ENTITY_KEY,
        tag=entity.FCT_ENTITY_TAG
    )

    file_artefact = args['some_file_artefact']

    # File upload
    session.file.upload(
        key=ab_beta_matrix.ENTITY.FCT_ENTITY_KEY,
        tag=ab_beta_matrix.ENTITY.FCT_ENTITY_TAG,
        data=csv,
        filename="file_name.csv"
    )

    # Save artefact value into Continuity
    artefact.ENTITY.FCT_ENTITY_DESC = "file_name.csv"
    artefact.ENTITY.IS_COMPLETE = True

    # Complete task
    session.update(entity, is_complete=True)
```

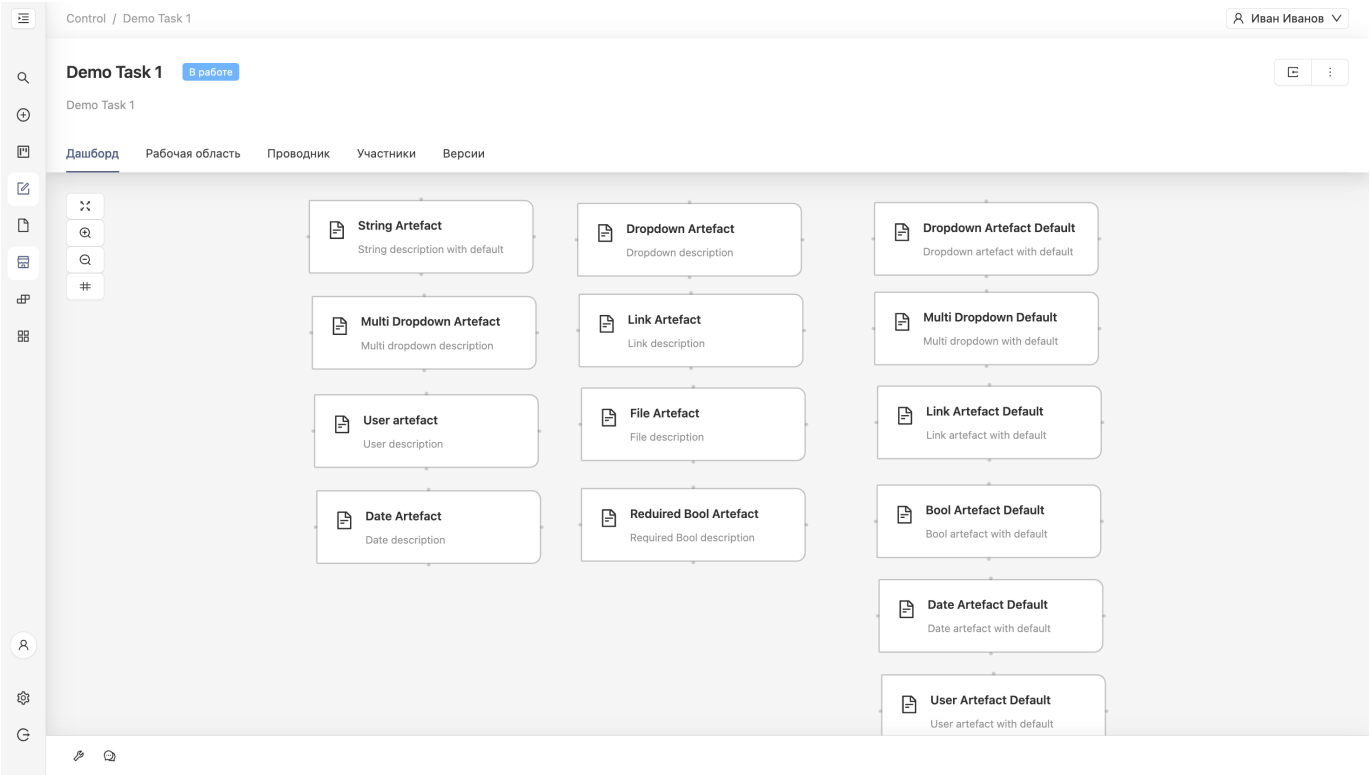
3.4.5 Шаблон в JSON представлении

Ниже представлены пример графа шаблона и его JSON-представление.

Основной workflow Demo-шаблона



Артефакты в первой задаче Demo-шаблона



JSON


```
{
    "DIM_ENTITY_KEY": "demo_docs_project",
    "DIM_ENTITY_TAG": "init",
    "DIM_ENTITY_NAME": "Demo project",
    "DIM_ENTITY_DESC": "Demo project",
    "ENTITY_TYPE": "project",
    "SWIMLANES": [
        {
            "SWIMLANE_NAME": "Бизнес-заказчик",
            "SWIMLANE_GROUPS": null,
            "IS_USED": true
        },
        {
            "SWIMLANE_NAME": "Аналитик",
            "SWIMLANE_GROUPS": null,
            "IS_USED": true
        }
    ],
    "PERMISSIONS": [
        {
            "PERMISSION": "owner",
            "GROUP": "writer"
        },
        {
            "PERMISSION": "owner",
            "GROUP": "writer"
        }
    ],
    "CONTEXTS": [],
    "EDGES": [
        {
            "SOURCE": "Task 1",
            "TARGET": "Task 2",
            "CONDITION": "\"{dropdown_artefact}\"==\"Value 1\"",
            "GROUP": null,
            "IS_BACKWARD": false
        },
        {
            "SOURCE": "Task 3",
            "TARGET": "Task 1",
            "CONDITION": "\"{final_artefact}\"==\"Value 2\"",
            "GROUP": null,
            "IS_BACKWARD": true
        },
        {
            "SOURCE": "Task 1",
```

```
            "TARGET": "Task 3",
            "CONDITION": "\"{dropdown_artefact}\"==\"Value 2\"",
            "GROUP": null,
            "IS_BACKWARD": false
        }
    ],
    "NODES": [
        {
            "ALIAS": "Task 1",
            "CONTEXTS": [
                {
                    "CONTEXT_KEY": "y",
                    "CONTEXT_VALUE": "-210",
                    "CONTEXT_LABEL": null
                },
                {
                    "CONTEXT_KEY": "x",
                    "CONTEXT_VALUE": "50",
                    "CONTEXT_LABEL": null
                }
            ],
            "ENTITY": {
                "DIM_ENTITY_KEY": "demo_docs_task_1",
                "DIM_ENTITY_TAG": "init",
                "DIM_ENTITY_NAME": "Demo Task 1",
                "DIM_ENTITY_DESC": "Demo Task 1",
                "ENTITY_TYPE": "task",
                "SWIMLANES": [
                    {
                        "SWIMLANE_NAME": "Аналитик",
                        "SWIMLANE_GROUPS": null,
                        "IS_USED": true
                    }
                ],
                "PERMISSIONS": [
                    {
                        "PERMISSION": "owner",
                        "GROUP": "writer"
                    }
                ],
                "CONTEXTS": [],
                "EDGES": [],
                "NODES": [
                    {
                        "ALIAS": "1",
                        "CONTEXTS": [
                            {
                                "CONTEXT_KEY": "x",
                                "CONTEXT_VALUE": "306",
                                "CONTEXT_LABEL": null
                            },
                            {
                                "CONTEXT_KEY": "y",
                                "CONTEXT_VALUE": "-313",
                                "CONTEXT_LABEL": null
                            }
                        ],
                        "ENTITY": {
                            "DIM_ENTITY_KEY": "string_artefact_docs",
                            "DIM_ENTITY_TAG": "init",
                            "DIM_ENTITY_NAME": "String Artefact",
                            "DIM_ENTITY_DESC": "String description with default",
                            "ENTITY_TYPE": "artefact",
                            "SWIMLANES": [],
                            "PERMISSIONS": [],
                            "CONTEXTS": [
                                {
                                    "CONTEXT_KEY": "type",
                                    "CONTEXT_VALUE": "string",
                                    "CONTEXT_LABEL": "String context description"
                                },
                                {
                                    "CONTEXT_KEY": "default",
                                    "CONTEXT_VALUE": "some_default_value",
                                    "CONTEXT_LABEL": "default_value_description"
                                }
                            ],
                            "EDGES": [],
                            "NODES": [],
                            "TAGS": []
                        }
                    },
                    {
                        "ALIAS": "10",
                        "CONTEXTS": [
                            {
                                "CONTEXT_KEY": "x",
                                "CONTEXT_VALUE": "666",
                                "CONTEXT_LABEL": null
                            },
                            {
                                "CONTEXT_KEY": "y",
                                "CONTEXT_VALUE": "-188",
                                "CONTEXT_LABEL": null
```

```
                }
            ],
            "ENTITY": {
                "DIM_ENTITY_KEY": "link_artefact_docs",
                "DIM_ENTITY_TAG": "init",
                "DIM_ENTITY_NAME": "Link Artefact",
                "DIM_ENTITY_DESC": "Link description",
                "ENTITY_TYPE": "artefact",
                "SWIMLANES": [],
                "PERMISSIONS": [],
                "CONTEXTS": [
                    {
                        "CONTEXT_KEY": "type",
                        "CONTEXT_VALUE": "link",
                        "CONTEXT_LABEL": "Link context description"
                    }
                ],
                "EDGES": [],
                "NODES": [],
                "TAGS": []
            }
        },
        {
            "ALIAS": "11",
            "CONTEXTS": [
                {
                    "CONTEXT_KEY": "y",
                    "CONTEXT_VALUE": "-54",
                    "CONTEXT_LABEL": null
                },
                {
                    "CONTEXT_KEY": "x",
                    "CONTEXT_VALUE": "313",
                    "CONTEXT_LABEL": null
                }
            ],
            "ENTITY": {
                "DIM_ENTITY_KEY": "user_artefact_docs",
                "DIM_ENTITY_TAG": "init",
                "DIM_ENTITY_NAME": "User artefact",
                "DIM_ENTITY_DESC": "User description",
                "ENTITY_TYPE": "artefact",
                "SWIMLANES": [],
                "PERMISSIONS": [],
                "CONTEXTS": [
                    {
                        "CONTEXT_KEY": "type",
                        "CONTEXT_VALUE": "user",
                        "CONTEXT_LABEL": "User context description"
                    }
                ],
                "EDGES": [],
                "NODES": [],
                "TAGS": []
            }
        },
        {
            "ALIAS": "12",
            "CONTEXTS": [
                {
                    "CONTEXT_KEY": "x",
                    "CONTEXT_VALUE": "669",
                    "CONTEXT_LABEL": null
                },
                {
                    "CONTEXT_KEY": "y",
                    "CONTEXT_VALUE": "-63",
                    "CONTEXT_LABEL": null
                }
            ],
            "ENTITY": {
                "DIM_ENTITY_KEY": "file_artefact_docs",
                "DIM_ENTITY_TAG": "init",
                "DIM_ENTITY_NAME": "File Artefact",
                "DIM_ENTITY_DESC": "File description",
                "ENTITY_TYPE": "artefact",
                "SWIMLANES": [],
                "PERMISSIONS": [],
                "CONTEXTS": [
                    {
                        "CONTEXT_KEY": "type",
                        "CONTEXT_VALUE": "file",
                        "CONTEXT_LABEL": "File context description"
                    }
                ],
                "EDGES": [],
                "NODES": [],
                "TAGS": []
            }
        },
        {
            "ALIAS": "13",
            "CONTEXTS": [
                {
```

```
                    "CONTEXT_KEY": "y",
                    "CONTEXT_VALUE": "74",
                    "CONTEXT_LABEL": null
                },
                {
                    "CONTEXT_KEY": "x",
                    "CONTEXT_VALUE": "316",
                    "CONTEXT_LABEL": null
                }
            ],
            "ENTITY": {
                "DIM_ENTITY_KEY": "date_artefact_docs",
                "DIM_ENTITY_TAG": "init",
                "DIM_ENTITY_NAME": "Date Artefact",
                "DIM_ENTITY_DESC": "Date description",
                "ENTITY_TYPE": "artefact",
                "SWIMLANES": [],
                "PERMISSIONS": [],
                "CONTEXTS": [
                    {
                        "CONTEXT_KEY": "type",
                        "CONTEXT_VALUE": "date",
                        "CONTEXT_LABEL": "Date context description"
                    }
                ],
                "EDGES": [],
                "NODES": [],
                "TAGS": []
            }
        },
        {
            "ALIAS": "14",
            "CONTEXTS": [
                {
                    "CONTEXT_KEY": "y",
                    "CONTEXT_VALUE": "71",
                    "CONTEXT_LABEL": null
                },
                {
                    "CONTEXT_KEY": "x",
                    "CONTEXT_VALUE": "669",
                    "CONTEXT_LABEL": null
                }
            ],
            "ENTITY": {
                "DIM_ENTITY_KEY": "bool_artefact_docs",
                "DIM_ENTITY_TAG": "init",
                "DIM_ENTITY_NAME": "Reduired Bool Artefact",
                "DIM_ENTITY_DESC": "Required Bool description",
                "ENTITY_TYPE": "artefact",
                "SWIMLANES": [],
                "PERMISSIONS": [],
                "CONTEXTS": [
                    {
                        "CONTEXT_KEY": "type",
                        "CONTEXT_VALUE": "bool",
                        "CONTEXT_LABEL": "Bull context description"
                    },
                    {
                        "CONTEXT_KEY": "require",
                        "CONTEXT_VALUE": "true",
                        "CONTEXT_LABEL": "Require context description"
                    }
                ],
                "EDGES": [],
                "NODES": [],
                "TAGS": []
            }
        },
        {
            "ALIAS": "15",
            "CONTEXTS": [
                {
                    "CONTEXT_KEY": "x",
                    "CONTEXT_VALUE": "1059.8897280966767",
                    "CONTEXT_LABEL": null
                },
                {
                    "CONTEXT_KEY": "y",
                    "CONTEXT_VALUE": "-310.15558912386706",
                    "CONTEXT_LABEL": null
                }
            ],
            "ENTITY": {
                "DIM_ENTITY_KEY": "dropdown_artefact_default_docs",
                "DIM_ENTITY_TAG": "init",
                "DIM_ENTITY_NAME": "Dropdown Artefact Default",
                "DIM_ENTITY_DESC": "Dropdown artefact with default",
                "ENTITY_TYPE": "artefact",
                "SWIMLANES": [],
                "PERMISSIONS": [],
                "CONTEXTS": [
                    {
                        "CONTEXT_KEY": "type",
```

```
                    "CONTEXT_VALUE": "dropdown",
                    "CONTEXT_LABEL": "Dropdown context description"
                },
                {
                    "CONTEXT_KEY": "value",
                    "CONTEXT_VALUE": "Value 1",
                    "CONTEXT_LABEL": "Value 1 context description"
                },
                {
                    "CONTEXT_KEY": "value",
                    "CONTEXT_VALUE": "Value 2",
                    "CONTEXT_LABEL": "Value 2 context description"
                },
                {
                    "CONTEXT_KEY": "default",
                    "CONTEXT_VALUE": "Value 2",
                    "CONTEXT_LABEL": "Default context description"
                }
            ],
            "EDGES": [],
            "NODES": [],
            "TAGS": []
        }
    },
    {
        "ALIAS": "16",
        "CONTEXTS": [
            {
                "CONTEXT_KEY": "x",
                "CONTEXT_VALUE": "1060.2009063444111",
                "CONTEXT_LABEL": null
            },
            {
                "CONTEXT_KEY": "y",
                "CONTEXT_VALUE": "-190.68882175226585",
                "CONTEXT_LABEL": null
            }
        ],
        "ENTITY": {
            "DIM_ENTITY_KEY": "multi_dropdown_default_docs",
            "DIM_ENTITY_TAG": "init",
            "DIM_ENTITY_NAME": "Multi Dropdown Default",
            "DIM_ENTITY_DESC": "Multi dropdown with default",
            "ENTITY_TYPE": "artefact",
            "SWIMLANES": [],
            "PERMISSIONS": [],
            "CONTEXTS": [
                {
                    "CONTEXT_KEY": "type",
                    "CONTEXT_VALUE": "dropdown",
                    "CONTEXT_LABEL": "Dropdown context description"
                },
                {
                    "CONTEXT_KEY": "multi",
                    "CONTEXT_VALUE": "true",
                    "CONTEXT_LABEL": "Multi context description"
                },
                {
                    "CONTEXT_KEY": "value",
                    "CONTEXT_VALUE": "Value 1",
                    "CONTEXT_LABEL": "Value 1 context description"
                },
                {
                    "CONTEXT_KEY": "value",
                    "CONTEXT_VALUE": "Value 2",
                    "CONTEXT_LABEL": "Value 2 context description"
                },
                {
                    "CONTEXT_KEY": "default",
                    "CONTEXT_VALUE": "Value 2",
                    "CONTEXT_LABEL": "Default context description"
                }
            ],
            "EDGES": [],
            "NODES": [],
            "TAGS": []
        }
    },
    {
        "ALIAS": "17",
        "CONTEXTS": [
            {
                "CONTEXT_KEY": "y",
                "CONTEXT_VALUE": "-65.53323262839879",
                "CONTEXT_LABEL": null
            },
            {
                "CONTEXT_KEY": "x",
                "CONTEXT_VALUE": "1064.0453172205434",
                "CONTEXT_LABEL": null
            }
        ],
        "ENTITY": {
            "DIM_ENTITY_KEY": "link_artefact_default_docs",
```

```
                "DIM_ENTITY_TAG": "init",
                "DIM_ENTITY_NAME": "Link Artefact Default",
                "DIM_ENTITY_DESC": "Link artefact with default",
                "ENTITY_TYPE": "artefact",
                "SWIMLANES": [],
                "PERMISSIONS": [],
                "CONTEXTS": [
                    {
                        "CONTEXT_KEY": "type",
                        "CONTEXT_VALUE": "link",
                        "CONTEXT_LABEL": "Link context description"
                    },
                    {
                        "CONTEXT_KEY": "default",
                        "CONTEXT_VALUE": "https://continuity-docs.k8s.datasapience.ru/user/template/template_about/",
                        "CONTEXT_LABEL": "Default context description"
                    }
                ],
                "EDGES": [],
                "NODES": [],
                "TAGS": []
            }
        },
        {
            "ALIAS": "18",
            "CONTEXTS": [
                {
                    "CONTEXT_KEY": "y",
                    "CONTEXT_VALUE": "66.31117824773415",
                    "CONTEXT_LABEL": null
                },
                {
                    "CONTEXT_KEY": "x",
                    "CONTEXT_VALUE": "1063.356495468278",
                    "CONTEXT_LABEL": null
                }
            ],
            "ENTITY": {
                "DIM_ENTITY_KEY": "bool_artefact_default_docs",
                "DIM_ENTITY_TAG": "init",
                "DIM_ENTITY_NAME": "Bool Artefact Default",
                "DIM_ENTITY_DESC": "Bool artefact with default",
                "ENTITY_TYPE": "artefact",
                "SWIMLANES": [],
                "PERMISSIONS": [],
                "CONTEXTS": [
                    {
                        "CONTEXT_KEY": "type",
                        "CONTEXT_VALUE": "bool",
                        "CONTEXT_LABEL": "Bool context description"
                    },
                    {
                        "CONTEXT_KEY": "default",
                        "CONTEXT_VALUE": "true",
                        "CONTEXT_LABEL": "Default true context description"
                    }
                ],
                "EDGES": [],
                "NODES": [],
                "TAGS": []
            }
        },
        {
            "ALIAS": "19",
            "CONTEXTS": [
                {
                    "CONTEXT_KEY": "y",
                    "CONTEXT_VALUE": "192.31117824773415",
                    "CONTEXT_LABEL": null
                },
                {
                    "CONTEXT_KEY": "x",
                    "CONTEXT_VALUE": "1066.045317220544",
                    "CONTEXT_LABEL": null
                }
            ],
            "ENTITY": {
                "DIM_ENTITY_KEY": "date_artefact_default_docs",
                "DIM_ENTITY_TAG": "init",
                "DIM_ENTITY_NAME": "Date Artefact Default",
                "DIM_ENTITY_DESC": "Date artefact with default",
                "ENTITY_TYPE": "artefact",
                "SWIMLANES": [],
                "PERMISSIONS": [],
                "CONTEXTS": [
                    {
                        "CONTEXT_KEY": "type",
                        "CONTEXT_VALUE": "date",
                        "CONTEXT_LABEL": "Date context description"
                    },
                    {
                        "CONTEXT_KEY": "default",
                        "CONTEXT_VALUE": "May 10 2023",
                        "CONTEXT_LABEL": "Default context description"
```

```
                }
            ],
            "EDGES": [],
            "NODES": [],
            "TAGS": []
        }
    },
    {
        "ALIAS": "2",
        "CONTEXTS": [
            {
                "CONTEXT_KEY": "y",
                "CONTEXT_VALUE": "-309",
                "CONTEXT_LABEL": null
            },
            {
                "CONTEXT_KEY": "x",
                "CONTEXT_VALUE": "664",
                "CONTEXT_LABEL": null
            }
        ],
        "ENTITY": {
            "DIM_ENTITY_KEY": "dropdown_artefact_docs",
            "DIM_ENTITY_TAG": "init",
            "DIM_ENTITY_NAME": "Dropdown Artefact",
            "DIM_ENTITY_DESC": "Dropdown description",
            "ENTITY_TYPE": "artefact",
            "SWIMLANES": [],
            "PERMISSIONS": [],
            "CONTEXTS": [
                {
                    "CONTEXT_KEY": "type",
                    "CONTEXT_VALUE": "dropdown",
                    "CONTEXT_LABEL": "Dropdown context desctiption"
                },
                {
                    "CONTEXT_KEY": "value",
                    "CONTEXT_VALUE": "Value 1",
                    "CONTEXT_LABEL": "Value 1 description"
                },
                {
                    "CONTEXT_KEY": "value",
                    "CONTEXT_VALUE": "Value 2",
                    "CONTEXT_LABEL": "Value 2 description"
                }
            ],
            "EDGES": [],
            "NODES": [],
            "TAGS": []
        }
    },
    {
        "ALIAS": "20",
        "CONTEXTS": [
            {
                "CONTEXT_KEY": "x",
                "CONTEXT_VALUE": "1069.5845133324574",
                "CONTEXT_LABEL": null
            },
            {
                "CONTEXT_KEY": "y",
                "CONTEXT_VALUE": "319.3111782477341",
                "CONTEXT_LABEL": null
            }
        ],
        "ENTITY": {
            "DIM_ENTITY_KEY": "user_artefact_default_docs",
            "DIM_ENTITY_TAG": "init",
            "DIM_ENTITY_NAME": "User Artefact Default",
            "DIM_ENTITY_DESC": "User artefact with default",
            "ENTITY_TYPE": "artefact",
            "SWIMLANES": [],
            "PERMISSIONS": [],
            "CONTEXTS": [
                {
                    "CONTEXT_KEY": "type",
                    "CONTEXT_VALUE": "user",
                    "CONTEXT_LABEL": "User context description"
                },
                {
                    "CONTEXT_KEY": "default",
                    "CONTEXT_VALUE": "writer",
                    "CONTEXT_LABEL": "Default context description"
                }
            ],
            "EDGES": [],
            "NODES": [],
            "TAGS": []
        }
    },
    {
        "ALIAS": "9",
        "CONTEXTS": [
            {
```

```
                        "CONTEXT_KEY": "x",
                        "CONTEXT_VALUE": "310",
                        "CONTEXT_LABEL": null
                    },
                    {
                        "CONTEXT_KEY": "y",
                        "CONTEXT_VALUE": "-185",
                        "CONTEXT_LABEL": null
                    }
                ],
                "ENTITY": {
                    "DIM_ENTITY_KEY": "multi_dropdown_artefact_docs",
                    "DIM_ENTITY_TAG": "init",
                    "DIM_ENTITY_NAME": "Multi Dropdown Artefact",
                    "DIM_ENTITY_DESC": "Multi dropdown description",
                    "ENTITY_TYPE": "artefact",
                    "SWIMLANES": [],
                    "PERMISSIONS": [],
                    "CONTEXTS": [
                        {
                            "CONTEXT_KEY": "type",
                            "CONTEXT_VALUE": "dropdown",
                            "CONTEXT_LABEL": "Dropdown context description"
                        },
                        {
                            "CONTEXT_KEY": "value",
                            "CONTEXT_VALUE": "Value 1",
                            "CONTEXT_LABEL": "Value 1 description"
                        },
                        {
                            "CONTEXT_KEY": "value",
                            "CONTEXT_VALUE": "Value 2",
                            "CONTEXT_LABEL": "Value 2 description"
                        },
                        {
                            "CONTEXT_KEY": "value",
                            "CONTEXT_VALUE": "Value 3",
                            "CONTEXT_LABEL": "Value 2 description"
                        },
                        {
                            "CONTEXT_KEY": "multi",
                            "CONTEXT_VALUE": "true",
                            "CONTEXT_LABEL": "Multi context description"
                        }
                    ],
                    "EDGES": [],
                    "NODES": [],
                    "TAGS": []
                }
            }
        ],
        "TAGS": []
    }
},
{
    "ALIAS": "Task 2",
    "CONTEXTS": [
        {
            "CONTEXT_KEY": "x",
            "CONTEXT_VALUE": "450",
            "CONTEXT_LABEL": null
        },
        {
            "CONTEXT_KEY": "y",
            "CONTEXT_VALUE": "-120",
            "CONTEXT_LABEL": null
        }
    ],
    "ENTITY": {
        "DIM_ENTITY_KEY": "demo_docs_task_2",
        "DIM_ENTITY_TAG": "init",
        "DIM_ENTITY_NAME": "Demo Task 2",
        "DIM_ENTITY_DESC": "Demo Task 2",
        "ENTITY_TYPE": "task",
        "SWIMLANES": [
            {
                "SWIMLANE_NAME": "Аналитик",
                "SWIMLANE_GROUPS": null,
                "IS_USED": true
            },
            {
                "SWIMLANE_NAME": "Бизнес-заказчик",
                "SWIMLANE_GROUPS": null,
                "IS_USED": true
            }
        ],
        "PERMISSIONS": [
            {
                "PERMISSION": "owner",
                "GROUP": "writer"
            }
        ],
        "CONTEXTS": [],
        "EDGES": [],
```

```
            "NODES": [],
            "TAGS": []
        }
    },
    {
        "ALIAS": "Task 3",
        "CONTEXTS": [
            {
                "CONTEXT_KEY": "x",
                "CONTEXT_VALUE": "450",
                "CONTEXT_LABEL": null
            },
            {
                "CONTEXT_KEY": "y",
                "CONTEXT_VALUE": "-310",
                "CONTEXT_LABEL": null
            }
        ],
        "ENTITY": {
            "DIM_ENTITY_KEY": "demo_docs_task_3",
            "DIM_ENTITY_TAG": "init",
            "DIM_ENTITY_NAME": "Demo Task 3",
            "DIM_ENTITY_DESC": "Demo Task 3",
            "ENTITY_TYPE": "task",
            "SWIMLANES": [
                {
                    "SWIMLANE_NAME": "Аналитик",
                    "SWIMLANE_GROUPS": null,
                    "IS_USED": true
                },
                {
                    "SWIMLANE_NAME": "Бизнес-заказчик",
                    "SWIMLANE_GROUPS": null,
                    "IS_USED": true
                }
            ],
            "PERMISSIONS": [
                {
                    "PERMISSION": "owner",
                    "GROUP": "writer"
                }
            ],
            "CONTEXTS": [],
            "EDGES": [],
            "NODES": [
                {
                    "ALIAS": "1",
                    "CONTEXTS": [
                        {
                            "CONTEXT_KEY": "y",
                            "CONTEXT_VALUE": "-309",
                            "CONTEXT_LABEL": null
                        },
                        {
                            "CONTEXT_KEY": "x",
                            "CONTEXT_VALUE": "664",
                            "CONTEXT_LABEL": null
                        }
                    ],
                    "ENTITY": {
                        "DIM_ENTITY_KEY": "final_artefact",
                        "DIM_ENTITY_TAG": "init",
                        "DIM_ENTITY_NAME": "Final Artefact",
                        "DIM_ENTITY_DESC": "Final description",
                        "ENTITY_TYPE": "artefact",
                        "SWIMLANES": [],
                        "PERMISSIONS": [],
                        "CONTEXTS": [
                            {
                                "CONTEXT_KEY": "type",
                                "CONTEXT_VALUE": "dropdown",
                                "CONTEXT_LABEL": "Dropdown context desctiption"
                            },
                            {
                                "CONTEXT_KEY": "value",
                                "CONTEXT_VALUE": "Value 1",
                                "CONTEXT_LABEL": "Value 1 description"
                            },
                            {
                                "CONTEXT_KEY": "value",
                                "CONTEXT_VALUE": "Value 2",
                                "CONTEXT_LABEL": "Value 2 description"
                            }
                        ],
                        "EDGES": [],
                        "NODES": [],
                        "TAGS": []
                    }
                }
            ],
            "TAGS": []
        }
    }
],
```

```
    "TAGS": []
}
```

4. Руководство пользователя A2P

4.1 О приложении

4.1.1 Функционал системы

A2P (далее Система) предоставляет набор инструментов для создания, настройки и продуктивизации моделей машинного обучения.

С помощью Системы Data Scientist (далее Пользователь) может проводить загрузку и обработку данных, разрабатывать модели, визуализировать результаты анализа и обеспечивать вывод разработанных моделей в эксплуатацию. Система позволяет тестировать и внедрять модели в промышленный контур (с помощью CI/CD процессов). Также предоставляется возможность контролировать и управлять жизненным циклом моделей машинного обучения.

Система обладает гибким функционалом и легкостью масштабирования вычислительных ресурсов и мониторинга. Для того, чтобы обеспечить пользователей необходимым набором инструментов, в систему включены ряд функциональных модулей и областей, среди которых основными являются:

- Инструменты анализа данных (на основе JupyterHub и JupyterLab);
- Среда исполнения моделей (на основе Airflow);
- Инструмент MLflow для логирования данных ML моделей (параметры, метрики, артефакты);
- Хранилище Minio S3 для нетекстовых (например, .pkl файлы) и/или больших по объему файлов;
- Репозиторий для кода модели и сопутствующих файлов в Gitlab.

Благодаря этим компонентам Пользователю предоставляются следующие возможности:

- Выбирать и запускать серверы Jupyter Notebook с заданным набором предустановленных программ, интерпретаторов, библиотек и фреймворков;
- Выбирать и запускать серверы Jupyter Notebook с заданным набором вычислительных ресурсов (количество виртуальных процессоров, объем памяти, расчет на CPU или GPU);
- Работать в привычной для Data Science специалиста среде Jupyter Notebook и автоматически сохранять разработанный код в репозиториях git;
- Проводить эксперименты с моделями машинного обучения; отслеживать результаты и метаданные с помощью удобных специализированных web-инструментов;
- Создавать и управлять жизненным циклом проектов машинного обучения;
- Упаковывать код в контейнер и размещать его в выделенном контуре для проверки работоспособности, тестирования, отладки и дальнейшего вывода в контур промышленной эксплуатации;
- Отслеживать ход выполнения рабочих процессов и получать доступ к сопутствующим артефактам.

4.2 Подготовка к работе

4.2.1 Уровень подготовки пользователя

Пользователь Системы должен являться специалистом в области обработки и анализа данных и иметь опыт работы с Jupyter Notebook с использованием браузера Google Chrome.

Также Пользователь должен обладать следующими знаниями:

- Знать соответствующую предметную область;
- Знать и иметь навыки работы с распределенной системой управления версиями Git;
- Иметь базовые знания работы в командной строке операционной системы Linux;
- Иметь представление о технологии контейнеризации Docker.

Квалификация пользователя должна позволять:

- Исследовать данные;
- Строить и проверять гипотезы;
- Проектировать и разрабатывать аналитические модели;
- Проводить обучение моделей, визуализировать результаты анализа с помощью языка Python.

4.2.2 Подготовка к работе

Перед началом работы следует убедиться, что на вашем компьютере установлен и исправно работает браузер Google Chrome (версии не ниже 104.0.5112.101/102).

Примечание: при использовании других браузеров возможны перебои и некорректная работа Системы.

Также необходимо проверить доступность сетевого подключения вашего компьютера. Для получения прав доступа к инструментам системы пользователь должен обратиться к Администратору Системы. Кроме того, Администратор системы должен сообщить Пользователю идентификатор группы в Gitlab. В системе принято разграничение прав в соответствии с ролевой моделью (см. «Ролевая модель»). Пользователь может узнать у Администратора, в какую из групп (ml-admin, ml-user, ml-suser, ml-auser) он входит.

4.2.3 Ролевая модель

OU-группы:

- ML-admins;
- ML-susers;
- ML-ausers;
- ML-users.

Tool	ML-admins	ML-susers	ML-ausers	ML-users
JupyterHub	1. Авторизация 2. Выбор ресурсов для контейнера CPU: 4, 8, 16, 24, 32; GPU: будет определен позднее; RAM: 4, 8, 16, 32, 64, 128, 256	1. Авторизация 2. Выбор ресурсов для контейнера CPU: 4, 8, 16, 24, 32; GPU: будет определен позднее; RAM: 4, 8, 16, 32, 64, 128, 256	1. Авторизация 2. Выбор ресурсов для контейнера CPU: 4, 8, 16; GPU: будет определен позднее; RAM: 4, 8, 16, 32, 64	1. Авторизация 2. Выбор ресурсов для контейнера CPU: 4, 8; GPU: будет определен позднее; RAM: 4, 8, 16
Airflow-dev (Инстанс Airflow для среды DEV)	1. Просмотр 2. Запуск	1. Просмотр 2. Запуск	1. Просмотр 2. Запуск	1. Просмотр 2. Запуск
Airflow-prod (Инстанс Airflow для среды PROD)	1. Просмотр 2. Запуск	1. Просмотр 2. Запуск	Просмотр	Просмотр
MLFlow	Полный доступ (просмотр, загрузка данных, создание экспериментов)	Полный доступ (просмотр, загрузка данных, создание экспериментов)	Полный доступ (просмотр, загрузка данных, создание экспериментов)	Полный доступ (просмотр, загрузка данных, создание экспериментов)
Minio	1. Авторизация 2. Создание и удаление бакетов 3. Создание и удаление политик 4. Изменение системных настроек	1. Авторизация 2. Чтение и запись в бакет A	1. Авторизация 2. Чтение и запись в бакет A	1. Авторизация 2. Чтение и запись в бакет A
Grafana	1. Просмотр 2.Редактирование	Просмотр	1. Просмотр 2.Редактирование	Просмотр
GitLab	На стороне Банка	На стороне Банка	На стороне Банка	На стороне Банка
Keycloak	Полный доступ	Нет доступа	Нет доступа	Нет доступа

4.3 Работа с приложением

4.3.1 Описание бизнес-логики использования A2P для разработки и продуктивизации batch ML-моделей

Данный раздел посвящен верхнеуровневому описанию действий Пользователя для создания и вывода в PROD модели машинного обучения. Предположим, что требуется разработать и вывести batch модель в PROD, чтобы эта модель делала предсказание (скоринг) по определенному расписанию (шедуленгу) и записывала результат в БД. Также пусть требуется обеспечить переобучение этой модели, но уже по другому расписанию.

Общий алгоритм работы в этом случае должен состоять из следующих этапов:

1. Создается проект в JupyterLab, содержащий необходимые шаблоны файлов и директорий. Внутри проекта можно создать Jupyter ноутбук, в котором вести написание и отладку модели. Более подробно про создание проекта и работу с JupyterLab смотрите в разделе “Работа с Jupyter”. Для разработки модели может потребоваться подключение к БД для получения данных. Если это подключение к Oracle, то подробнее см. пункт “Подключение к БД Oracle”. Если требуется подключение к Hive, то подробнее см. пункт “Подключение к БД Hive”.
   Также при обучении модели желательно логировать процесс обучения в MLflow (см. п. «Работа с MLFlow»).
   Результатом обучения модели должен стать файл model.pkl, находящийся в директории pkl внутри текущего проекта. Кроме файла model.pkl Пользователь должен подготовить файлы main.py, retrain.py (папка models), dag-batch.py и dag-retrain.py (папка gitlab-ci). Более подробно про назначение, особенности и необходимое содержание файлов смотрите в разделах “Файл main.py”, “Файл retrain.py”, “Файл dag-batch.py”, “Файл dag-retrain.py”, также следует ознакомиться с пунктом “Передача переменных в python скрипты”.
   Подводя итог, разработку модели можно считать завершенной, когда готовы файлы model.pkl, main.py, retrain.py, dag-batch.py, dag-retrain.py, если использовались вновь установленные библиотеки, то еще файл requirements.txt.
2. Отправить pickle файл с моделью в удаленное хранилище S3 Minio с помощью команд dvc, более подробно смотрите пункт “Работа с dvc”,
3. С помощью команд git загрузить (“запушить”) подготовленные файлы модели в ветку dev удаленного репозитория. Более подробно смотрите пункт “Работа с git”.
4. Необходимо перейти в GitLab и запустить CI/CD процесс продуктивизации модели в среде dev. Для этого перейти в удаленный репозиторий и выполнить слияние ветки dev с веткой main. Более подробно смотрите в пункте “Работа с GitLab”.
5. Необходимо перейти в airflow и активировать DAG файл с моделью, который и будет выполнять скоринг модели по заданному в DAG-файле расписанию (шедуленгу).
6. Далее необходимо запустить CI/CD пайплайн для переобучения модели. Для этого необходимо выполнить слияние ветки main с веткой retrain. Более подробно смотрите в пункте “CI/CD процесс для переобучения модели”.
7. Перейти в airflow и активировать DAG-файл с переобучением модели. После этого, согласно расписанию, указанному в данном DAG-файле, модель будет проходить переобучение. Более подробно смотрите в пункте “CI/CD процесс для переобучения модели”.
8. Для перевода модели из среды dev в среду prod требуется выполнить слияние ветки main в ветку prod. Более подробно смотрите в пункте “CI/CD процесс для продуктивизации модели в среде PROD”.

Модель успешно разработана и поставлена на регламентное исполнение!

4.3.2 Описание операций

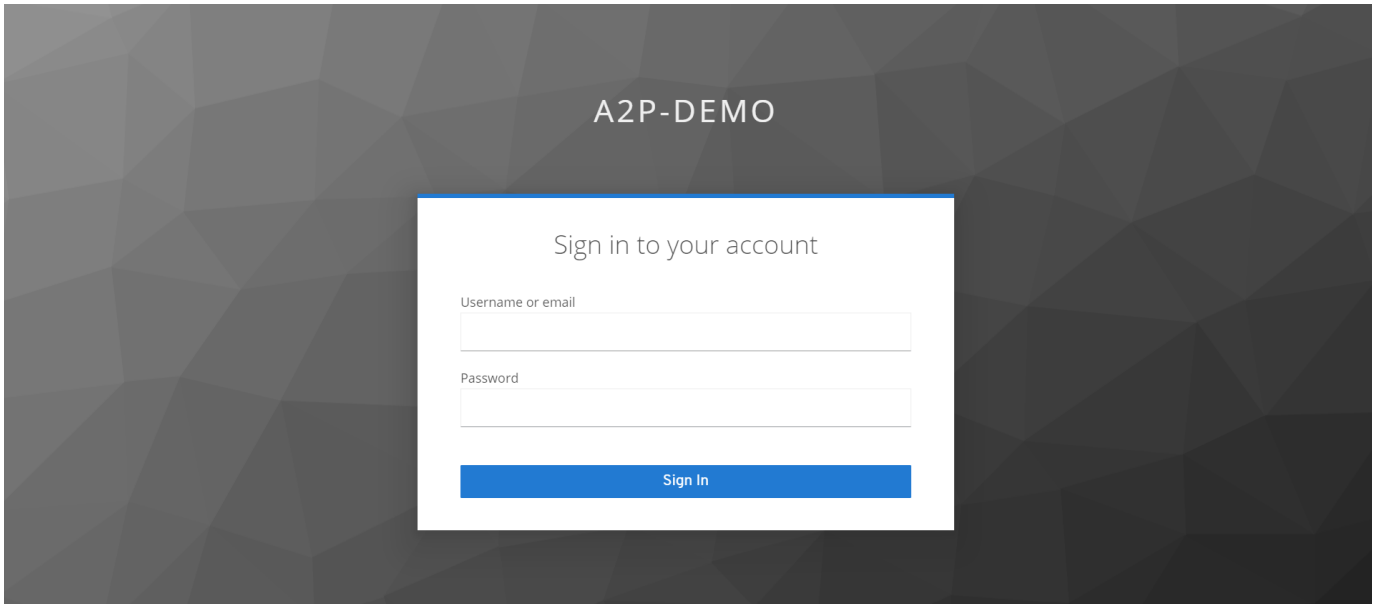
Описание операций

РАБОТА С JUPYTER

Авторизация в системе

В адресной строке браузера необходимо ввести ссылку, полученную от Администратора Системы для доступа в модуль A2P.

При первоначальном входе в Систему Пользователю будет предоставлено окно авторизации. Для успешного входа в связанные между собой приложения используется инструмент «Keycloak», он позволяет провести аутентификацию пользователя единожды и получить доступ к другим приложениям без последующего входа в каждом из них, кроме GitLab.

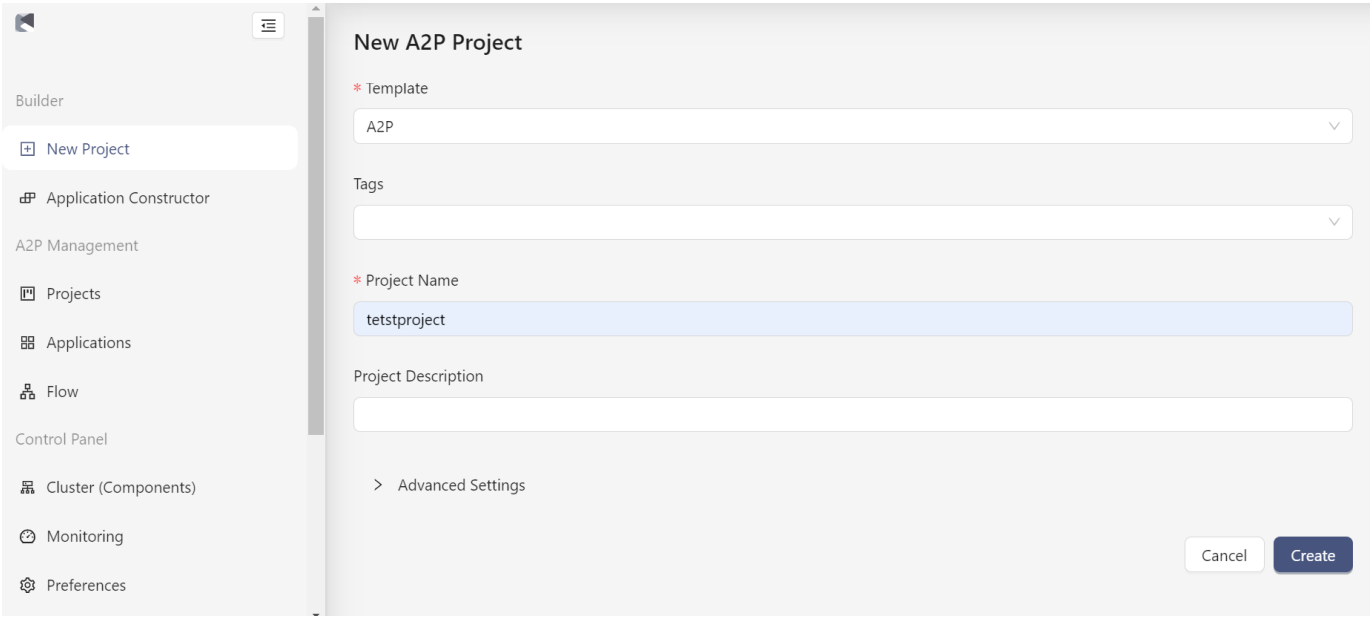


Примечание: настройки групп пользователей уточняйте у Администратора Системы

Создание проекта в A2P

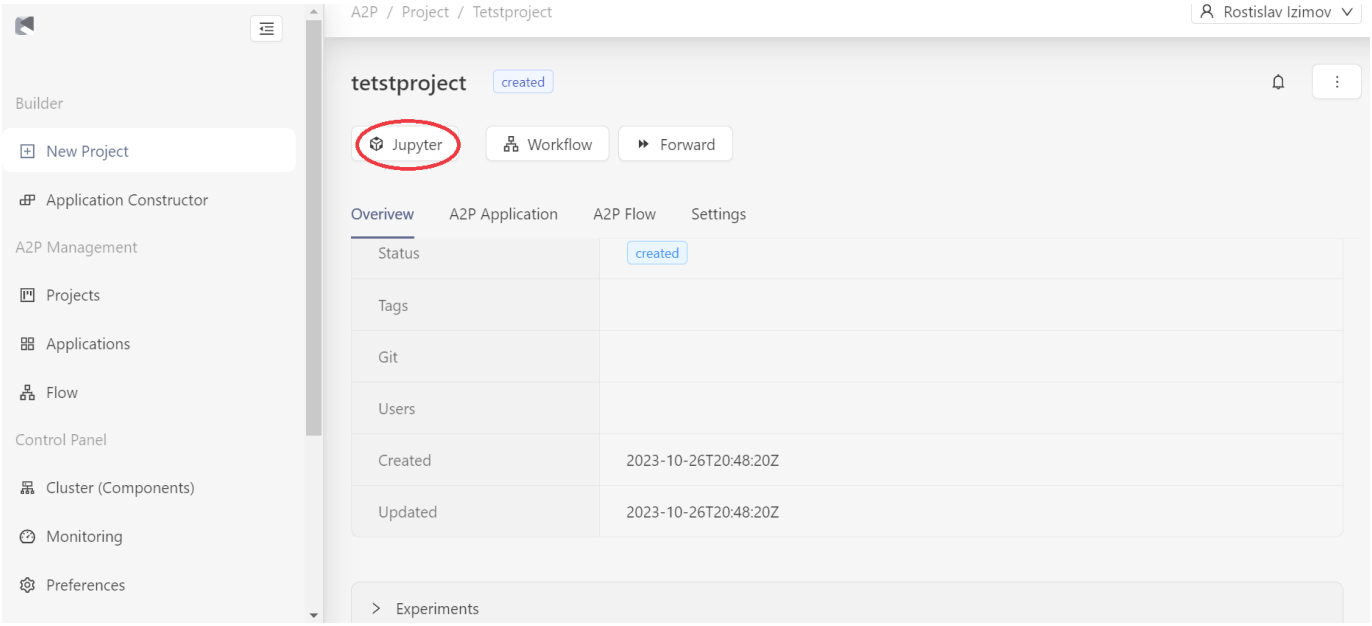
Для создания проекта необходимо нажать бна "New Project", затем задать параметры проекта (см. рисунок ниже):

- Выбрать шаблон (Template);
- Указать тэги (Tags);
- Ввести имя проекта (Project Name);
- Добавить описание проекта при необходимости (Priject Description).



Инициализация Jupyter контейнера

После создания проекта в модуле A2P на странице с информацией о новом проекта необходимо перейти в Jupyter (см. рисунок ниже).

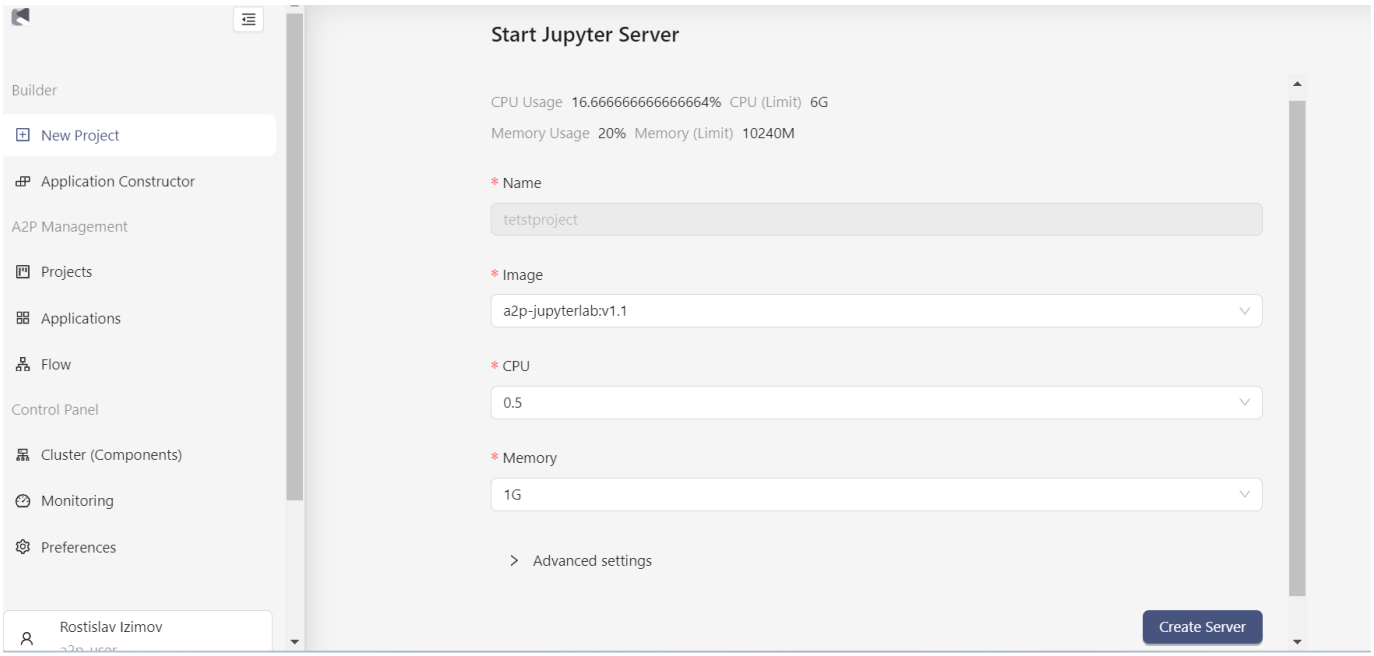


Выбор и запуск JupyterLab сервера

В открывшемся окне необходимо выбрать желаемый стек и вычислительные параметры Jupyter сервера.

В системе предусмотрен выбор базового образа. Если разрабатываемая модель машинного обучения не подразумевает работы с нейронными сетями, то следует использовать первый более “легкий” образ, в противном случае выбирайте второй - более “тяжелый”.

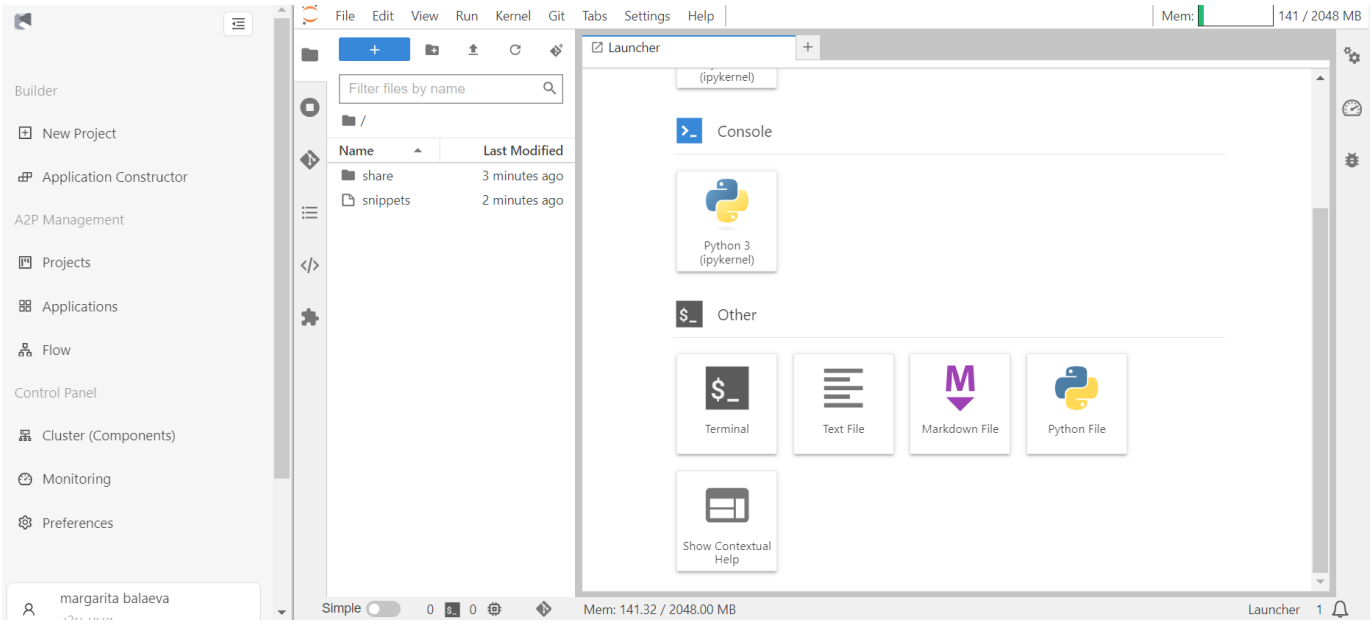
Полный список предустановленных в каждый образ библиотек можно узнать у Администратора. Следует помнить, что если Пользователю дополнительно нужны новые библиотеки, то он может их установить с помощью терминала и команды `pip install`.



Стек (преднастроенное пользовательское окружение) можно выбрать из выпадающего списка. Аналогичным образом задаются другие параметры: количество ядер и размер оперативной памяти сервера.

После того как параметры выбраны, нажмите на кнопку “Create Server”. Это запустит процесс создания сервера Jupyter Lab.

После создания сервера происходит автоматическое перенаправление на основной экран JupyterLab.



Примечание: время жизни сервера неограниченно, поэтому после завершения работы требуется обязательно завершить сессию, во избежание частичной потери информации. Для завершения сессии перейдите в меню “File”, выберите “Hub Contol Panel”, в открывшемся окне выберите “Stop My Server”. В ряде случаев возможна принудительная остановка сервера Администратором после определенного времени бездействия Пользователя.

Структура проекта для batch модели

Проект представляет собой способ организации и описания кода, который принят в рамках рассматриваемой Системы. Единый подход к организации кода облегчает совместную работу нескольких специалистов, а также открывает возможности использования средств автоматизации.

Проект представляет собой структуру файлов и директорий, которая также является локальной версией репозитория git. В директории проекта могут находиться различные файлы, включая файлы блокнотов, скрипты, файлы конфигурации.

Структура проекта для batch модели:

```
├── README.md - Файл с описанием бизнес назначения модели
├── data - Директория для хранения датасетов
| ├── processed
| └── raw
├── gitlab-ci - Директория, содержащая конфигурационные файлы
| ├── Dockerfile - докерфайл для продуктивизации модели
| ├── dag-batch.py - DAG файл для применения модели в Airflow
| ├── dag-retrain.py - DAG файл для вызова переобучения в Airflow
| ├── mlflow-create-model.py - Python скрипт для работы с MLflow
| ├── mlflow-stage-model.py - Python скрипт для работы с MLflow
| └── start.sh - файл для bash, запускаемый в контейнере
├── logging - Директория для логирования работы модели или иных логов
├── media - Директория для медиа файлов, например картинок .png.
├── models
| ├── main.py - Python скрипт для скоринга моделью
| └── retrain.py - Python скрипт для переобучения модели
├── notebooks - Директория для Jupyter ноутбуков
├── online - Директория с файлами для онлайн модели
| ├── Dockerfile
| ├── deployment.yaml
| ├── ingress.yaml
| ├── requirements.txt
| ├── service.yaml
| └── src
| ├── api.py
| ├── html
| | └── index.html
| ├── pkl
| ├── scheme_api
| | └── scheme.py
| └── solver
| └── main.py
├── pkl - Директория для хранения pkl файлов
└── requirements.txt Файл с необходимыми библиотеками
```

Примечание: описание используемых моделью библиотек должно находиться в файле requirements.txt, который располагается в основной папке проекта. Для создания файла можно выполнить в терминале (см. раздел Работа с терминалом) команду pip freeze > requirements.txt, при этом нужно убедиться, что полученный файл оказался в корневой директории проекта. Данный файл можно оставить пустым, если Пользователь не устанавливал никаких дополнительных библиотек к уже существующим в данном базовом образе. Более подробно про дополнительные библиотеки смотрите в разделе "Установка дополнительных библиотек".

Структура проекта для online модели

Структура проекта для online модели:

```
├── README.md - Файл с описанием бизнес назначения модели
├── chart - Директория для файлов helm-chart
| ├── Chart.yaml
| ├── NOTES.txt
| ├── templates
| | ├── _helpers.tpl
```

```
| | ├── deployment.yaml
| | ├── hpa.yaml
| | ├── ingress.yaml
| | ├── service.yaml
| | └── serviceaccount.yaml
| └── values.yaml
├── gitlab-ci - Директория, содержащая конфигурационные файлы
| ├── Online-Dockerfile - докерфайл для продуктивизации online модели
| ├── Online-start.sh
| ├── Retrain-Dockerfile - докерфайл для продуктивизации retrain модели
| ├── Retrain-start.sh
| ├── dag-retrain.py - DAG файл для вызова переобучения в Airflow
| ├── mlflow-create-model.py - Python скрипт для работы с MLflow
| └── mlflow-stage-model.py - Python скрипт для работы с MLflow
├── notebooks
├── requirements.txt - Файл с необходимыми библиотеками
├── sample-model - Директория с примером обучения шаблонной online модели
| ├── online_notebook.ipynb
| └── sample.json - Пример входного вектора для шаблонной online модели
├── sonar-project.properties - Файл с параметрами Sonarqube
└── src
├── api.py - Файл с api веб-сервиса на fastapi
├── html
| └── index.html - Файл с html страницей для шаблонной online модели
├── pkl - Директория для сохранения сериализованных файлов модели
├── scheme_api
| └── scheme.py
└── solver
├── main.py - Скрипт для применения online модели
└── retrain.py - Скрипт для переобучения online модели
```

Описание файла main.py

Файл предназначен для скоринга обученной ранее моделью. После создания проекта файл main.py создается по шаблону и имеет вид, представленный ниже:

```
1     import os
2     import pandas as pd
3     import pickle
4
5     print("TARGET_DATE=", os.environ.get('TARGET_DATE'))
6
7     target_date = os.environ.get('TARGET_DATE', None)
8
9     if target_date is None:
10        raise RuntimeError("Error! Не передана дата скоринга!")
11
12    date_for_prediction = target_date
13
14    with open("./pkl/model.pkl","rb") as f:
15        p = pickle.load(f)
16        print(p)
17    print("good")
```

На 5 строке происходит вывод на экран переменной окружения TARGET_DATE, которая считывается в скрипт main.py с помощью команды os.environ.get. Данная переменная далее должна использоваться как дата для скоринга моделью. Если данная переменная окружения не обнаружена, то работа скрипта будет прервана исключением на строке 10.

Далее на 14 строке происходит открытие *.pkl файла с ранее обученной моделью, после этого данная модель может быть использована для скоринга.

Пользователь должен модифицировать данный файл со своими потребностями. Как правило, данный файл должен содержать получение переменных для скоринга (как, например, заданная в шаблоне переменная TARGET_DATE), затем открытие ранее обученной модели в формате .pkl, далее подключение к необходимой БД и загрузку данных, на

которых нужно выполнить скоринг, далее непосредственно скоринг и, наконец, запись предсказанных значений в БД. Для подключения к БД потребуется использовать соответствующие учетные данные, более подробно смотрите в пунктах данного Руководства Пользователя про подключение к БД, а для передачи переменных можно посмотреть пункт «Передача переменных в Python скрипты».

Описание файла retrain.py

Файл retrain.py предназначен для переобучения модели. Также он может быть использован для первоначального обучения модели на этапе разработки модели в Jupyter. Шаблонный файл retrain.py представлен на ниже:

```
1     import pickle
2     import os
3
4     print("DATE_FOR_TRAIN=", os.environ.get('DATE_FOR_TRAIN'))
5     print("DATE_FOR_VALIDATION=", os.environ.get('DATE_FOR_VALIDATION'))
6
7     date_for_train = os.environ.get('DATE_FOR_TRAIN', None)
8     date_for_validation = os.environ.get('DATE_FOR_VALIDATION', None)
9
10    if (date_for_train is None) or (date_for_validation is None):
11        raise RuntimeError("Error! Не передана дата для обучения или для валидации!")
12
13
14
15    data = {"version": 1}
16
17    with open('./pkl/model.pkl', 'wb') as f:
18        pickle.dump(data, f)
```

В шаблоне заложено считывание из переменных окружения среды, где запускается данный файл двух дат: для обучения (DATE_FOR_TRAIN) и валидации (DATE_FOR_VALIDATION) (строки 7 и 8). Если данные даты отсутствуют в среде, где запускается данный файл, то работа скрипта будет прервана (строки 10–11). Далее в шаблоне на строке 15 обозначается “заглушка” вместо реальной модели, которая потом сохраняется как model.pkl. Пользователь должен модифицировать данный файл в соответствии со своими потребностями, общий подход, может быть, например, следующим. В начале файла в данный скрипт передаются необходимые переменные как переменные окружения среды, где запускается данный скрипт (более подробно смотрите пункт «Передача переменных в Python скрипты»). Например, это могут быть даты для обучения, даты для валидации (как в шаблоне), данные учетной записи для подключения к БД. Далее может идти блок подключения к БД, загрузка данных и непосредственно код обучения модели. Обучение модели следует логировать в MLflow (более подробно про логирование в MLflow смотрите пункт документа «Логирование эксперимента»). После того, как объект модели обучен, он должен быть сохранен в папку pkl с названием model.pkl.

Описание файла dag-batch.py

Файл dag-batch.py отвечает за создание DAG-файла для Airflow, который отвечает за скоринг моделью, то есть за запуск скрипта main.py.

Этот файл содержит следующие важные параметры:

- указываются необходимые потребные ресурсы для продуктивизированного контейнера.
- указываются какие секреты Кубернетес будет примонтированы к продуктивизированному контейнеру. В шаблонном файле реализовано примонтирование секрета с keytab-файлом для аутентификации в Hive через kerberos и секрет tech-ora-creds с данными технической учетной записи для подключения к Oracle.
- задается расписание запуска модели (schedule).
- задаются переменные окружения, которые будут переданы в скрипт main.py, запускаемый в продуктивизированном контейнере.

Рассмотрим шаблон dag-batch.py ниже:

```
1     from airflow import DAG
2     from datetime import datetime, timedelta, date
3     from airflow.contrib.operators.kubernetes_pod_operator import KubernetesPodOperator
4     from kubernetes.client import models as k8s
5     import pytz
6    # Connection via tech-creds
7    ora_creds_vars  = [k8s.V1EnvFromSource(secret_ref=k8s.V1SecretEnvSource(name='tech-ora-creds'))]
8
9
10    target_date = date.today().strftime('%d.%m.%Y')
```

```
11
12    default_args = {
13        'owner': 'airflow',
14        'depends_on_past': False,
15        'start_date': datetime(2020, 11, 1, tzinfo=pytz.timezone('Europe/Moscow')),
16        'email': ['${TP_ML_USER}@glowbyteconsulting.com'],
17        'email_on_failure': False,
18        'email_on_retry': False,
19        'retries': 1,
20        'retry_delay': timedelta(minutes=5)
21    }
22
23    dag = DAG(
24      '${CI_PROJECT_NAME}',
25      default_args=default_args,
26      # schedule_interval='*/15 * * * *',
27      schedule_interval = timedelta(hours=12),
28      catchup=False,
29      params = {'target_date':target_date}
30      )
31
32    container_resources=k8s.V1ResourceRequirements(
33                                          requests={'cpu': '1', 'memory': '1Gi',
                                                    'nvidia.com/gpu': '0'},
34                                          limits={'cpu': '2', 'memory': '4Gi',
                                                    'nvidia.com/gpu': '0'},
35                                          )
36
37    passing = KubernetesPodOperator(
38      namespace='${NS}',
39      image="${DOCKER_REGISTRY}/${NEXUS_SPACE}/${CI_PROJECT_NAME}:${TAG}.${CI_PIPELINE_ID}",
40      name="${CI_PROJECT_NAME}_${CI_PIPELINE_ID}",
41      task_id="run_model",
42      volume_mounts=[volume_mount],
43      volumes=[volume],
44      get_logs=True,
45      dag=dag,
46      resources=container_resources,
47      is_delete_operator_pod=True,
48      env_from=ora_creds_vars,
49      env_vars= [k8s.V1EnvVar(name='TARGET_DATE', value="{{ params.target_date }}")]
50      )
51
52    passing
```

Отметим, что в рассматриваемом примере шаблона файла, строки, начинающиеся с $ и далее слова, обернутого в фигурные скобки являются переменными, которые при создании проекта (create_project) заполняются соответствующими именами. Например, Пользователь при создании проекта указывает имя проекта, пусть testproject, тогда в данном случае строки 23-30 будут заполнены следующим образом:

```
23    dag = DAG(
24      'testproject',
25      default_args=default_args,
26      # schedule_interval='*/15 * * * *',
27      schedule_interval = timedelta(hours=12),
28      catchup=False,
29      params = {'target_date':target_date}
30      )
```

В 7 строке из секрета с именем tech-ora-creds забираются переменные окружения для технической учетной записи для Oracle, имена этих переменных перечислены в Приложении Б «Список имен переменных окружения и их значения для секрета tech-ora-creds». Далее данные переменные окружения передаются в контейнер с моделью, это указано на строке 48.

В 10 строке обозначается переменная target_date, которой присваивается значение текущего дня.

В 12–21 строках указываются default аргументы для объекта DAG, следует обратить внимание на параметр retries и retray_delay. В случае, если в Airflow данный DAG-файл при попытке запуска получил ошибку, то данные параметры отвечают за число повторных запусков и за задержку между ними. Соответственно, при шаблонных значениях, в случае ошибки при старте, данный DAG-файл попробует перезапуститься еще 1 раз через 5 минут после первого запуска.

В 23–30 строках обозначается объект DAG, здесь стоит обратить внимание на параметр schedule_interval, отвечающий за запуск DAG-файла по расписанию. Допускается несколько форматов задания расписания, например, можно реализовывать расписание с помощью cron-шаблона (для удобства использования данного формата можно сначала отладить нужное расписание с помощью сервиса https://crontab.guru/, а затем вставить его как значение расписания. Либо можно задавать расписание с помощь объекта timedelta из библиотеки datetime. Именно такой метод расписания указан в шаблонном файле, таким образом следующий запуск данного DAG-файла будет спустя 12

часов после первого запуска. Третий способ задания расписания запуска – это с помощью конструкции Timetable Airflow, более подробную информацию по такому способу задания расписания можно посмотреть на официальном сайте Airflow). Также в 29 строке указывается переменная params, в которую передается словарь (dict), содержащий название и значение переменной (в случае данного шаблона - target_date).

В строках 32–35 задаются параметры ресурсов (CPU, RAM, GPU) для продуктивизированного контейнера, то есть контейнера, в котором будет запускаться скрипт main.py. Более подробно про выбор ресурсов смотрите в разделе “Выбор ресурсов для продуктивизированного контейнера“.

В 37–50 строках обозначается объект KubernetesPodOperator, с помощью которого средствами Airflow будет запускаться контейнер с продуктивизированной моделью. Важно обратить внимание на строку 48, где происходит передача переменных окружения из секрета с технической учетной записи для Oracle и на строку 49, где также передается переменная окружения, но не из секрета, а обозначенная в данном файле на строке 10 target_date. Параметр name внутри конструкции на данной строке обозначает имя, под которым данная переменная будет передана в контейнер, то есть это имя переменной окружения, которая будет доступна при запуске скрипта main.py в случае обращения к ней с помощью метода os.environ.get(‘TARGET_DATE’). Параметр value содержит специальную конструкцию, обернутую в двойные фигурные кавычки, которая позволяет передать переменной TARGET_DATE значение, присвоенное на строке 10. Кроме того, если данный DAG-файл в Airflow будет запущен с опцией “run DAG w config”, где Пользователем будет указано иное значение target_date, то в контейнер со скриптом main.py будет переданное новое значение, а не то, что было присвоено на строке 10.

Пользователь Системы должен модифицировать данный файл под свои требования при разработке модели, например, указывать нужное расписание и передавать нужные переменные с необходимыми значениями в контейнер.

Описание файла dag-retrain.py

Файл dag-retrain.py отвечает за создание DAG-файла для Airflow, который отвечает за переобучение модели, то есть за запуск скрипта retrain.py.

Этот файл содержит следующие важные параметры:

- указываются необходимые потребные ресурсы для продуктивизированного контейнера.
- указываются даты для обучения (тренинга) и валидации модели.
- указываются какие секреты Kubernetes будет примонтированы к продуктивизированному контейнеру с переобучением. В шаблонном файле реализовано примонтирование секрета с keytab-файлом для аутентификации в Hive через kerberos и секрет tech-ora-creds с данными технической учетной записи для подключения к Oracle.
- задается расписание запуска переобучения модели (schedule).
- задаются переменные окружения, которые будут переданы в скрипт retrain.py, запускаемый в продуктивизированном контейнере.

Рассмотрим шаблон dag-retrain.py ниже:

```
1     from airflow import DAG
2     from datetime import datetime, timedelta, date
3     from airflow.contrib.operators.kubernetes_pod_operator import KubernetesPodOperator
4     from kubernetes.client import models as k8s
5     import pytz
6
7     #Для переобучения модели нужно указать дату для обучения модели и дату для валидации.
8     #Их можно задать как таймдельту от текущего дня, либо передавать это параметрами конфига,
9     #с которым запускается dag-файл.
10    date_for_train = (date.today() - timedelta(weeks=4)).strftime('%d.%m.%Y')
11    date_for_validation = (date.today() - timedelta(weeks=2)).strftime('%d.%m.%Y')
12
13
14    volumes = [
15        k8s.V1Volume(
16        name='retrain-volume',
17    persistent_volume_claim=k8s.V1PersistentVolumeClaimVolumeSource(claim_name='${CI_PIPELINE_ID}-retrain-tmp-dir'),)
18    ]
19
20   volume_mounts =
21        k8s.V1VolumeMount(mount_path='/app/pkl', name='retrain-volume')
22    ]
23
24    # Connection via tech-creds
25    ora_creds_vars = [k8s.V1EnvFromSource(secret_ref=k8s.V1SecretEnvSource(name='tech-ora-creds'))]
26
```

```
    default_args = {
        'owner': 'airflow',
        'depends_on_past': False,
        'start_date': datetime(2020, 11, 1, tzinfo=pytz.timezone('Europe/Moscow')),
        'email': ['${TP_ML_USER}@glowbyteconsulting.com'],
        'email_on_failure': False,
        'email_on_retry': False,
        'retries': 1,
        'retry_delay': timedelta(minutes=5)
    }

    env_vars = [
    k8s.V1EnvVar(name='CI_SERVER_HOST', value='$CI_SERVER_HOST'),
    k8s.V1EnvVar(name='GIT_GROUP', value='$GIT_GROUP'),
    k8s.V1EnvVar(name='CI_PROJECT_NAME', value='$CI_PROJECT_NAME'),
    k8s.V1EnvVar(name='CI_PIPELINE_ID', value='$CI_PIPELINE_ID'),
    k8s.V1EnvVar(name='S3_BUCKET', value='$S3_BUCKET'),
    k8s.V1EnvVar(name='S3_ENDPOINTURL', value='$S3_ENDPOINTURL'),
    k8s.V1EnvVar(name='NS', value='$NS')
    ]

    env_from_list = [k8s.V1EnvFromSource(secret_ref=k8s.V1SecretEnvSource(name='retrain-creds'))]

    dag = DAG(
      '${CI_PROJECT_NAME}_retrain',
      default_args=default_args,
      # schedule_interval='*/15 * * * *',
      schedule_interval = timedelta(minutes=60),
      catchup=False,
      params = {'date_for_train':date_for_train, 'date_for_validation':date_for_validation}
      )

    container_resources=k8s.V1ResourceRequirements(
                                          requests={'cpu': '1', 'memory': '1Gi'},
                                          limits={'cpu': '2', 'memory': '4Gi'},
                                          )

    pvc_create = KubernetesPodOperator(
      namespace='${NS}',
      image="${DOCKER_REGISTRY}/${NEXUS_SPACE}/${TECH_IMAGE}",
      name="pvc-create-${CI_PROJECT_NAME}-${CI_PIPELINE_ID}",
      service_account_name="retrain-sa",
      task_id="pvc_create",
      cmds=["bash"],
      arguments=["sh/gen_pvc.sh"],
      env_vars=env_vars,
      env_from=env_from_list,
      get_logs=True,
      dag=dag,
    is_delete_operator_pod=True,
      )

    # Retrain

    retrain = KubernetesPodOperator(
      namespace='${NS}',
      image="${DOCKER_REGISTRY}/${NEXUS_SPACE}/${CI_PROJECT_NAME}:${TAG}.${CI_PIPELINE_ID}",
      name="retrain-${CI_PROJECT_NAME}-${CI_PIPELINE_ID}",
      task_id="retrain_model",
      volume_mounts=volume_mounts,
      volumes=volumes,
      get_logs=True,
      dag=dag,
      resources=container_resources,
      env_from=ora_creds_vars+env_from_list,
      env_vars={"DATE_FOR_TRAIN": "{{ params.date_for_train }}",
                "DATE_FOR_VALIDATION": "{{ params.date_for_validation }}"},
      is_delete_operator_pod=True,
      )

   push_pkl = KubernetesPodOperator(
      namespace='${NS}',
      image="${DOCKER_REGISTRY}/${NEXUS_SPACE}/${TECH_IMAGE}",
      name="push-pkl-${CI_PROJECT_NAME}-${CI_PIPELINE_ID}",
      task_id="push_pkl",
      cmds=["bash"],
      arguments=["sh/git_dvc.sh"],
    # arguments=["sh/git_dvc_f.sh"],
      env_vars=env_vars,
      env_from=env_from_list,
      volume_mounts=volume_mounts,
      volumes=volumes,
      get_logs=True,
      dag=dag,
      is_delete_operator_pod=True,
      )

    del_pvc = KubernetesPodOperator(
      namespace='${NS}',
      image="${DOCKER_REGISTRY}/${NEXUS_SPACE}/${TECH_IMAGE}",
      name="del-pvc-${CI_PROJECT_NAME}-${CI_PIPELINE_ID}",
      service_account_name="retrain-sa",
      task_id="del_pvc",
      cmds=["bash"],
```

```
121      arguments=["sh/del_pvc.sh"],
122      env_vars=env_vars,
123      env_from=env_from_list,
124      get_logs=True,
125      dag=dag,
126      is_delete_operator_pod=True,
127      )
128
129    retrain.set_upstream(pvc_create)
130    push_pkl.set_upstream(retrain)
131    del_pvc.set_upstream(push_pkl)
```

Отметим, что в рассматриваемом примере шаблона файла, строки, начинающиеся с $ и далее слова, обернутого в фигурные скобки являются переменными, которые при создании проекта (creat_project) заполняются соответствующими именами. Например, Пользователь при создании проекта указывает имя проекта, пусть testproject, тогда в данном случае строка 41 будет такой:

```
17 k8s.V1EnvVar(name='CI_PROJECT_NAME', value='testproject'),
```

Рассмотрим основные функции кода в данном файле. На 10–11 строках обозначаются переменные `date_for_train` и `date_for_validation`, отвечающие за дату для обучения и для валидации, необходимые для скрипта переобучения. Их значения берутся как сегодняшняя дата минус 4 недели и минус 2 недели соответственно.

Пользователь при разработке реальной модели должен изменить данные даты по своему усмотрению.

На 25 строке обозначается переменная ora_creds_vars, значением которой является список переменных окружения из технической учетной записи для Oracle, список и значения данных переменных можно найти в Приложении Б «Список имен переменных окружения и их значения для секрета tech-ora-creds».

На 27–36 строках указываются default аргументы для объекта DAG, следует обратить внимание на параметр retries и retray_delay. В случае, если в Airflow данный DAG-файл при попытке запуска получил ошибку, то данные параметры отвечают за число повторных запусков и за задержку между ними. Соответственно, при шаблонных значениях, в случае ошибки при старте, данный DAG-файл попробует перезапуститься еще 1 раз через 5 минут после первого запуска.

Строки 38–48 задают необходимые технические переменные окружения.

На 50–57 строках обозначается объект DAG, здесь стоит обратить внимание на параметр schedule_interval, отвечающий за запуск DAG-файла по расписанию. Допускается несколько форматов задания расписания, например, можно реализовывать расписание с помощью cron-шаблона (для удобства использования данного формата можно сначала отладить нужное расписание с помощью сервиса https://crontab.guru/, а затем вставить его как значение расписания. Либо можно задавать расписание с помощь объекта timedelta из библиотеки datetime). Именно такой метод расписания указан в шаблонном файле, таким образом следующий запуск данного DAG-файла будет спустя 12 часов после первого запуска. Третий способ задания расписания запуска – это с помощью конструкции Timetable Airflow, более подробную информацию по такому способу задания расписания можно посмотреть на официальном сайте Airflow. Также на 56 строке указывается переменная params, в которую передается словарь (dict), содержащий названия и значения переменных.

На строках 59–62 задаются ресурсы контейнера, в котором будет выполняться переобучение модели, то есть ресурсы контейнера, где будет запускаться скрипт retrain.py. Более подробно про выбор ресурсов контейнера смотрите в разделе “Выбор ресурсов для продуктивизированного контейнера“.

Далее обозначаются четыре объекта KubernetesPodOperator, первый, третий и четвертый являются техническими, поэтому их описание можно пропустить. Более подробно рассмотрим второй KubernetesPodOperator (строки 81–95). Важно обратить внимание на строки 91 и 92, на которых передаются в контейнер, где запускается скрипт переобучения retrain.py, заданные ранее из секретов или в параметрах (params) переменные окружения.

Передача переменных в python скрипты

Для эффективной работы в Системе Пользователь должен понимать как реализована передача переменных для python скриптов, запускаемых в контейнере.

Предположим, что разработана модель, имеется файл main.py, в котором запускается обученная модель (файл model.pkl), далее в этом скрипте необходимо обратиться в БД, чтобы получить данные для скоринга. В этом случае дата (target_date), на которую нужно получить, является переменной, которую нужно передать в скрпит main.py,

который будет в контейнере по расписанию, указанному в файле dag-batch.py. То есть, когда произойдет активация DAG-файла в Airflow, то будет запущен контейнер, в котором фактически будет выполнена команда python3 main.py. Рассмотрим механизм передачи переменной target_date внутрь скрипта main.py. Значение target_date должно быть указано в файле dag-batch.py, например, так:

```
from datetime import date
target_date = date.today()
```

Далее в файле dag-batch.py эта переменная должна быть указана в params объекта dag:

```
dag = DAG(
  '${CI_PROJECT_NAME}',
  default_args=default_args,
  # schedule_interval='*/15 * * * *',
  schedule_interval = timedelta(hours=12),
  catchup=False,
  params = {'target_date':target_date}
  )
```

Здесь в params передается объект типа словарь (dict), где в качестве ключа (key) выступает имя переменной, а в качестве значения (value) само значение переменной. Наконец, в этом же файле dag-batch.py при создании объекта KubernetesPodOperator указать в параметрах env_vars (пример ниже).

```
passing = KubernetesPodOperator(
  namespace='${NS}',
  image="${DOCKER_REGISTRY}/${NEXUS_SPACE}/${CI_PROJECT_NAME}:${TAG}.${CI_PIPELINE_ID}",
  name="${CI_PROJECT_NAME}_${CI_PIPELINE_ID}",
  task_id="run_model",
  volume_mounts=[volume_mount],
  volumes=[volume],
  get_logs=True,
  dag=dag,
  is_delete_operator_pod=True,
  env_from=ora_creds_vars,
  env_vars= [k8s.V1EnvVar(name='TARGET_DATE', value="{{ params.target_date }}")]
  )
```

Рассмотрим более подробно строку:

```
 env_vars= [k8s.V1EnvVar(name='TARGET_DATE', value="{{ params.target_date }}")]
```

В качестве значения env_vars присваивается список, в данном примере содержащий единственный элемент. Этот элемент является объектом `k8s.V1EnvVar` , с помощью которого можно задать переменную окружения, которая будет доступна в контейнере с python скриптом. У объекта `k8s.V1EnvVar` необходимо задать параметр `name`, который задает имя данной переменной окружения и параметр `value` , который задает значение данной переменной окружения. Значение параметра `value` задается с помощью специальной конструкции в двойных фигурных скобках, где само значение берется из params объекта dag, рассмотренного выше.

Теперь при запуске данного DAG'а в Airflow внутри контейнера, где будет запущен скрипт main.py будет доступная переменная окружения TARGET_DATE, имеющая значение сегодняшней даты. Для того, чтобы внутри скрипта main.py обратиться к данной переменной необходимо выполнить следующий код:

```
import os
target_date = os.environ.get('TARGET_DATE')
```

Следует отметить, что Пользователь на этапе разработке модели должен продумывать какие переменные нужно передавать для разрабатываемой им модели и модифицировать файл main.py и dag-batch.py под свои потребности.

Полностью аналогичный механизм передачи переменных реализован и для переобучения модели (файл retrain.py и dag-retrain.py), только для файла dag-retrain.py следует обратить внимание, что передача переменных окружения проводится во втором объекте KubernetesPodOperator (всего в файле dag-retrain.py четыре объекта KubernetesPodOperator).

В заключение данного раздела следует отметить следующее. На этапе разработки модели (напомним, разработка модели идет в контейнере для разработки, то есть в продакшине модель работает в "продуктовом" контейнере, а разрабатывается в контейнере для разработки) важно отладить работу скриптов main.py и retrain.py. При этом, если скрипты main.py и retrain.py содержат обращения к переменным окружения с помощью os.environ.get, то необходимо предварительно обозначить эти переменные как переменные окружения в терминале, из которого

запускаются данные скрипты. Например, если требуется отладить внутри контейнера скрипт main.py, который обращается к переменной окружения os.environ.get('TARGET_DATE'), то в Jupyter надо открыть терминал (более подробно про работу с терминалом смотрите в разделе «Работа с терминалом»), в нем указать команду export TARGET_DATE="01.01.2022", далее в данном же терминале перейти в папку с проектом и запускать скрипт main.py. В этом случае после строки в main.py `target_date = os.environ.get('TARGET_DATE')` значением переменной target_date будет строка "01.01.2022".

Похожий прием с экспортом переменных окружения перед запуском скриптов python в терминале может быть использован для обозначения переменных, связанных с учетными записями для доступа к БД, которые необходимы при отладке.

Выбор ресурсов для продуктивизированного контейнера

При разработке batch модели в файлах dag-batch.py и dag-retrain.py необходимо указать ресурсы контейнера, в котором будет производиться скоринг моделью (запуск файла main.py) и переобучение модели (запуск файла retrain.py). Под ресурсами здесь понимается количество ядер CPU и объем RAM.

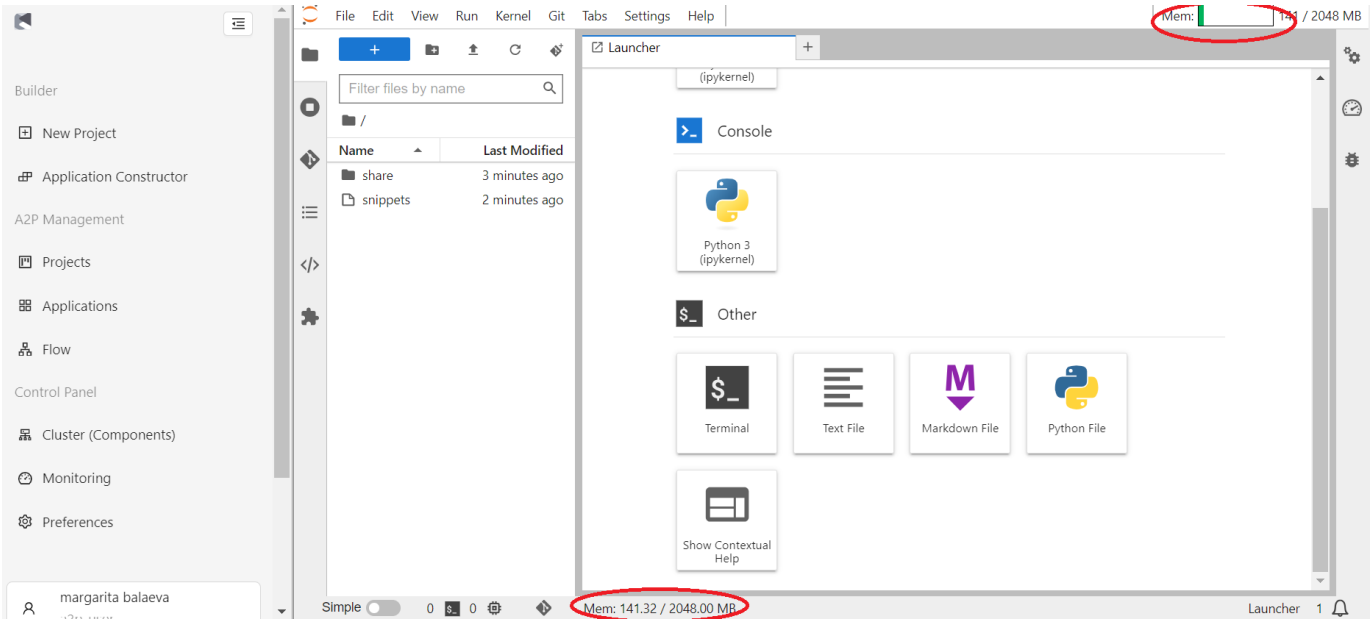
В параметрах контейнера могут быть указаны два параметра, задающие параметры ресурсов, с которыми запустить контейнер:

- limits - верхняя граница ресурсов, если работа контейнера начнет потреблять больше ресурсов, чем данное значение, то контейнер будет остановлен средствами Kubernetes.
- requests - нижняя граница ресурсов, необходимых для работы контейнера. Например, если мы знаем, что для работы контейнера с моделью, которая делает скоринг нужно не менее 30гб RAM, то должны указать это в requests. В этом случае, когда данный DAG-файл будет активирован в Airflow, то Kubernetes будет искать ноду кластера, на которой имеется указанное в requests количество ресурсов. Если такой ноды найдено не будет, то Kubernetes не сможет запустить контейнер и в логах Airflow появиться ошибка о том, что pod с моделью не смог запуститься. В этом случае следует обратиться к Администратору системы. Напомним, синтаксис задания ресурсов должен быть следующий:

```
container_resources = k8s.V1ResourcesRequirements(
                                                limits={'cpu':'2',   'memory':'4Gi','nvidia.com/gpu':'0'},
                                                requests={'cpu':'1', 'memory':'1Gi', 'nvidia.com/gpu':'0'}}
                                                )
```

  В этом примере минимальный набор ресурсов для контейнера будет 1 CPU, 1Гб оперативной памяти и ни одного GPU, а верхний лимит, при выходе за который Kubernetes "убьет", контейнер составляет 2 CPU и 4Гб оперативной памяти. В приведенном примере количество GPU указано равным нулю, данное значение берется при создании Проекта модели в creat_project.ipynb на этапе вопроса про необходимость использования библиотеки CUDA. То есть, если на том этапе указано значение, например, 1, то оно будет подставлено в файлы dag-batch.py и dag-retrain.py. Обратите внимание, если для работы скоринга модели и переобучения модели требуется использование GPU, то необходимо указывать количество требуемых ядер GPU.

При разработке модели возникает вопрос о том, какие же границы ресурсов выбрать. Для определения параметров контейнера с моделью, то есть контейнера для скоринга моделью, а не контейнера переобучения, на этапе разработки модели в Jupyter запустить скрипт main.py и посмотреть сколько памяти потребляется при работе скрипта, рисунок Ресурсы Jupyter.



Взяв данное максимальное значение потребляемой памяти во время работы скрипта, можно увеличить его на 10–20% и использовать это число как в requests. Число CPU для requests можно взять равное тому значению, которое было выбрано при старте контейнера с Jupyter. Далее верхнюю границу ресурсов (limits) можно выбрать такое же как requests. Для контейнера с переобучение модели выбор ресурсов может быть выполнен по аналогичной схеме, только наблюдать за потребляемой памятью нужно во время запуска скрипта retrain.py.

Для онлайн моделей значения необходимых ресурсов задается в файле online/deployment.yaml на строке “resources”. Здесь также нужно задать количество cpu и memory для полей requests и limits. Если для работы онлайн модели требуется GPU, то необходимо ниже строки “memory” добавить строку nvidia.com/gpu: "1", где вместо 1 может быть необходимое число ядер GPU. Пример файла deployment с GPU можно посмотреть в ветке online проекта gbc/hlkupliftmodel.

Установка дополнительных библиотек

Образ, который выбран и запущен в начале работы, уже содержит определенный набор библиотек и фреймворков списком которых можно запросить у Администратора. У Пользователя существует возможность установить в рамках сервера дополнительные библиотеки.

Для установки библиотек Python можно использовать команду `pip install`.

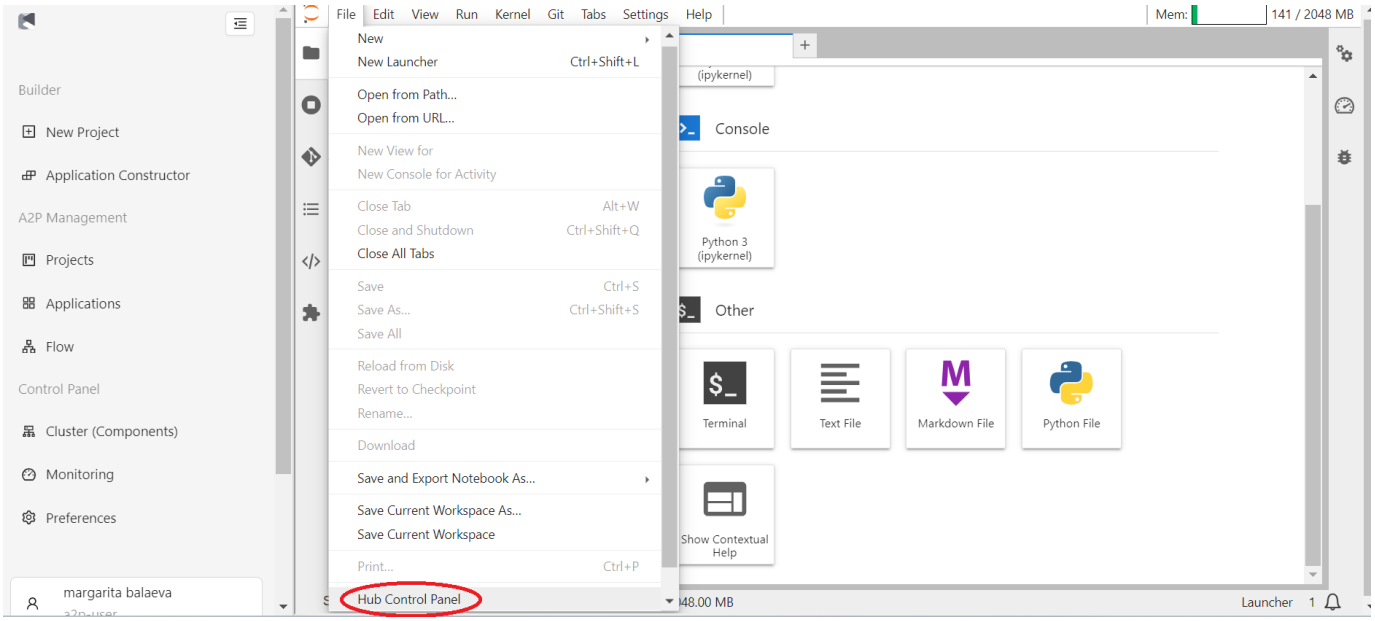
Для вызова `pip` нужно сначала перейти в режим терминала (см. раздел Работа с терминалом) или запустить эту программу непосредственно из блокнота (см. соответствующий раздел документации IPython). Следует отметить, что установленные Пользователем библиотеки сохраняются в персистивное хранилище данного Пользователя, поэтому будут доступны после удаления (остановки), текущего Jupyter сервера, иными словами, достаточно один раз установить недостающую библиотеку, а при последующих запусках JupyterLab данная библиотека будет уже доступна, однако, чтобы данная свежеустановленна библиотека была доступна при последующей продуктивизации модели ее название следует добавить в файл requirements.txt.

Если потребуется добавить библиотеку в базовый стек (базовый образ), чтобы она была доступна всем пользователям системы, то обратитесь к Администратору.

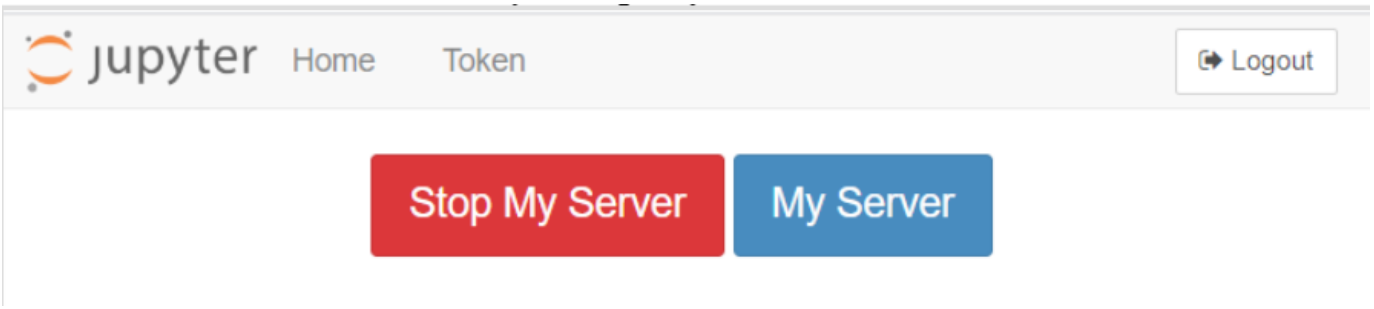
Завершение работы с JupyterLab сервером

В случае если требуется завершить текущий сеанс работы над проектом модели, необходимо сохранить внесенные в код изменения и остановить работу контейнера, чтобы освободить ресурсы системы. Для этого:

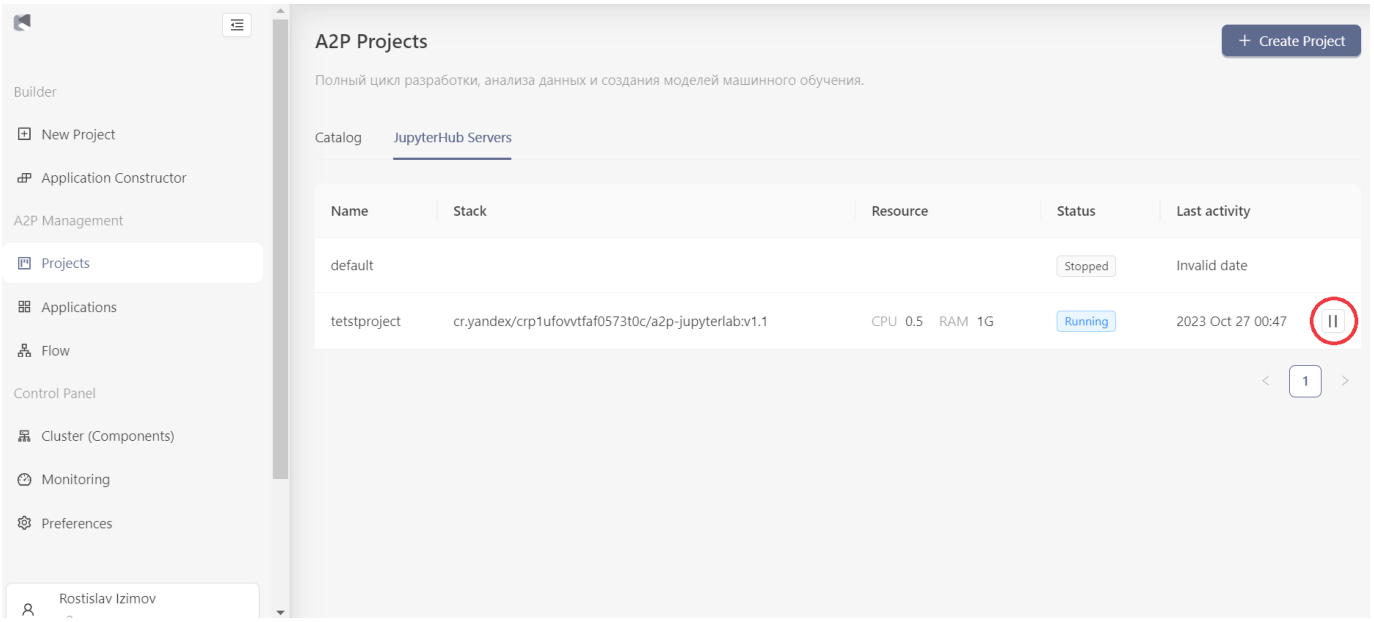
- Находясь внутри окна блокнота, следует нажать на меню **“File”** и далее **“Save notebook”** в левом верхнем углу;
- Перейдите в раздел “File” основного меню и выберите пункт **“Hub Control Panel”**



В открывшемся окне нажмите на кнопку **“Stop My Server”**.



Также завершить работу с сервером можно из модуля A2P. Для этого необходимо перейти в "Projects", далее в "JupyterHub Servers", выбрать знак паузы у интересующего контейнера (см. рисунок).



Описание операций

РАБОТА С MLFLOW

Общие сведения

Управление экспериментами и жизненным циклом моделей реализовано с помощью open source решения MLflow.

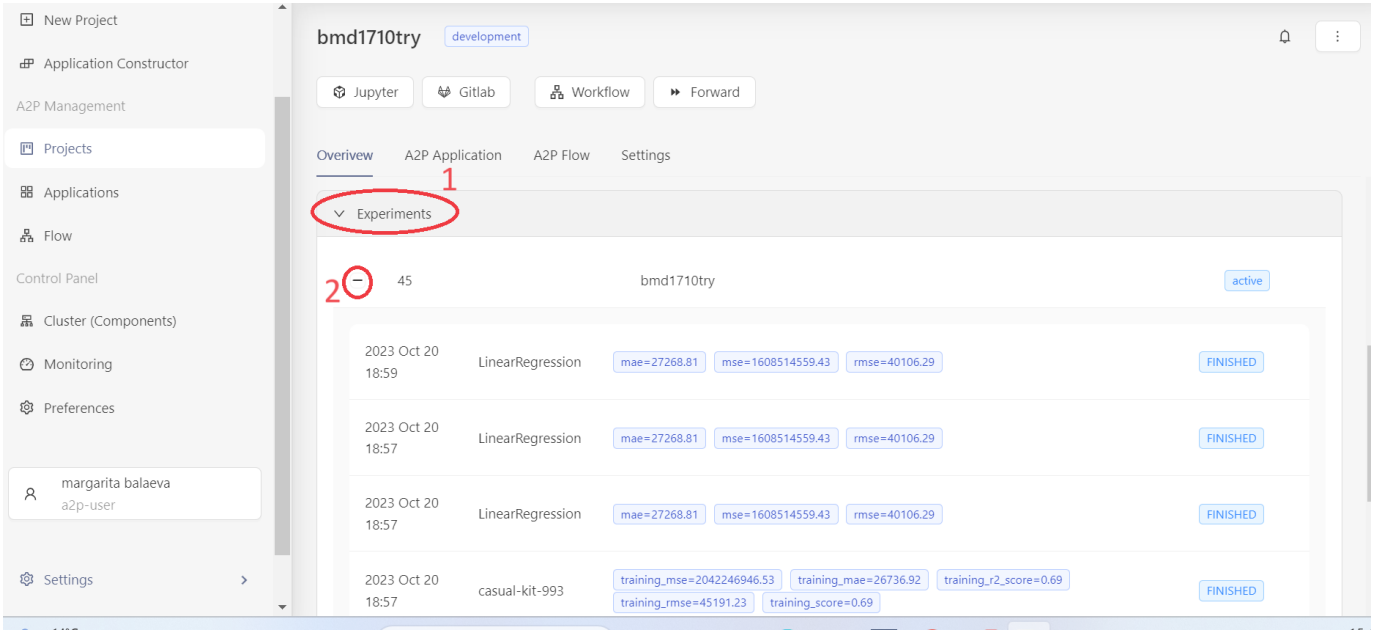
С помощью MLflow Пользователь системы имеет возможность при каждом запуске кода сохранять и отслеживать следующие данные:

- Параметры модели (это значения параметров, при которых данная модель обучалась, например, для решающего дерева максимальная глубина дерева является параметром);
- Метрики модели (это показатели качества уже обученной модели, например, ROC-AUC);
- Теги эксперимента/модели - информация (например: имя разработчика, ссылка на используемую витрину) об особенности конкретного запуска/модели;
- Имя файла, который вызывает запуск кода;
- Версию кода;
- Различные артефакты, например изображения, файлы моделей, файлы в формате .md (для описания задачи и модели, решающей ее);
- Модель (сохраняется в качестве артефакта) - папка model с файлом модели в формате .pkl.

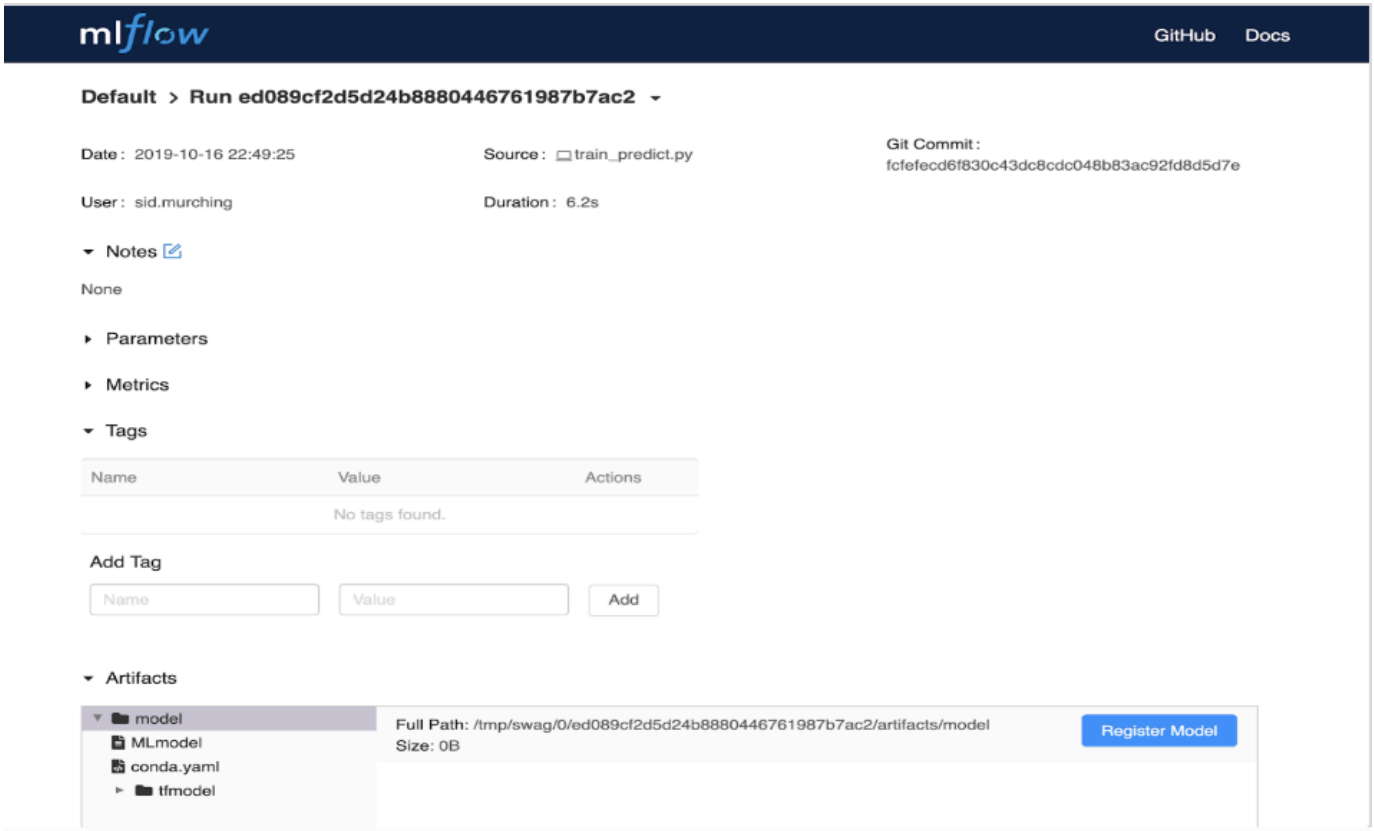
После завершения выполнения кода Пользователь имеет возможность посмотреть собранные данные с помощью веб-интерфейса MLFlow.

Просмотр экспериментов в MLFlow

Для просмотра реестра экспериментов проекта необходимо перейти во вкладку **Experiments** на странице проекта в модуле A2P, далее развернуть запуски эксперимента с интересующими метриками нажатием на "+" (см. рисунок ниже).



Для перехода на страницу с подробной информацией о конкретном запуске (run) необходимо кликнуть на запись соответвующего запуска.



Логирование эксперимента

Рассмотрим логирование на примере обучения и предсказания простейшей модели решающего дерева, код которого представлен ниже:

```
1  import mlflow
2  import numpy as np
3  from sklearn.tree import DecisionTreeRegressor
4  from sklearn.model_selection import train_test_split
5  from sklearn.linear_model import LinearRegression
6  from sklearn.metrics import *
7
8  X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.5, random_state=5)
9
10 experiment_name = 'Test_experiment'
11 mlflow.set_experiment(f"{experiment_name}")
12 mlflow.set_experiment_tag("project", "")
13 for i, max_depth in enumerate([5, 10, 30], start=1):
14     with mlflow.start_run(run_name=f'DecisionTree_run_{i}') as run:
15         dtr = DecisionTreeRegressor(max_depth = max_depth,
16                                     min_samples_leaf = 5,
17                                     ccp_alpha = 0.00018)
18
19         dtr = dtr.fit(X_train, y_train)
20
21         y_pred = dtr.predict(X_test)
22         mae = mean_absolute_error(y_test, y_pred)
23         mse = mean_squared_error(y_test, y_pred)
24         rmse = np.sqrt(mean_squared_error(y_test, y_pred))
25
26         mlflow.log_metric('mae', mae)
27         mlflow.log_metric('mse', mse)
28         mlflow.log_metric('rmse', rmse)
29         mlflow.log_param('model', 'DecTree')
30         mlflow.log_param('max_depth', max_depth)
```

На строке 8 происходит разделение датасетов с фичами X и таргетом Y на данные для тренировки и валидации, следует обратить внимание, что предварительно X и Y необходимо загрузить отдельно.

На строках 10–11 задается имя эксперимента, под которым он будет отображаться в интерфейсе MLflow на вкладке Experiments. В случае, если ранее в Системе такого эксперимента ранее не было, то на экран будет выведено уведомление о создании эксперимента с данным именем. Рекомендуется создавать отдельный эксперимент для каждой модели.

На строке 12 задается тег экмперимента.

На строках 13-30 задан цикл, который проходит одновременно по индексу i от 1 до 3 и по параметру max_depth (максимальная глубина решающего дерева) по значениям 5, 10, 30. Внутри каждого цикла на строке 14 с помощью контекстного менеджера with выполняется запуск рана (run). Под раном подразумевается участок кода, имеющий отношение к ML-модели. Имя рана в данном случае будет DecisionTree_run_i, где i принимает значения 1, 2, 3.

На строке 15 обозначается создание объекта решающего дерева, которому в качестве параметра максимальной глубины передается значение max_depth, равное 5, 10 и 30.

На 19 строке происходит обучение дерева.

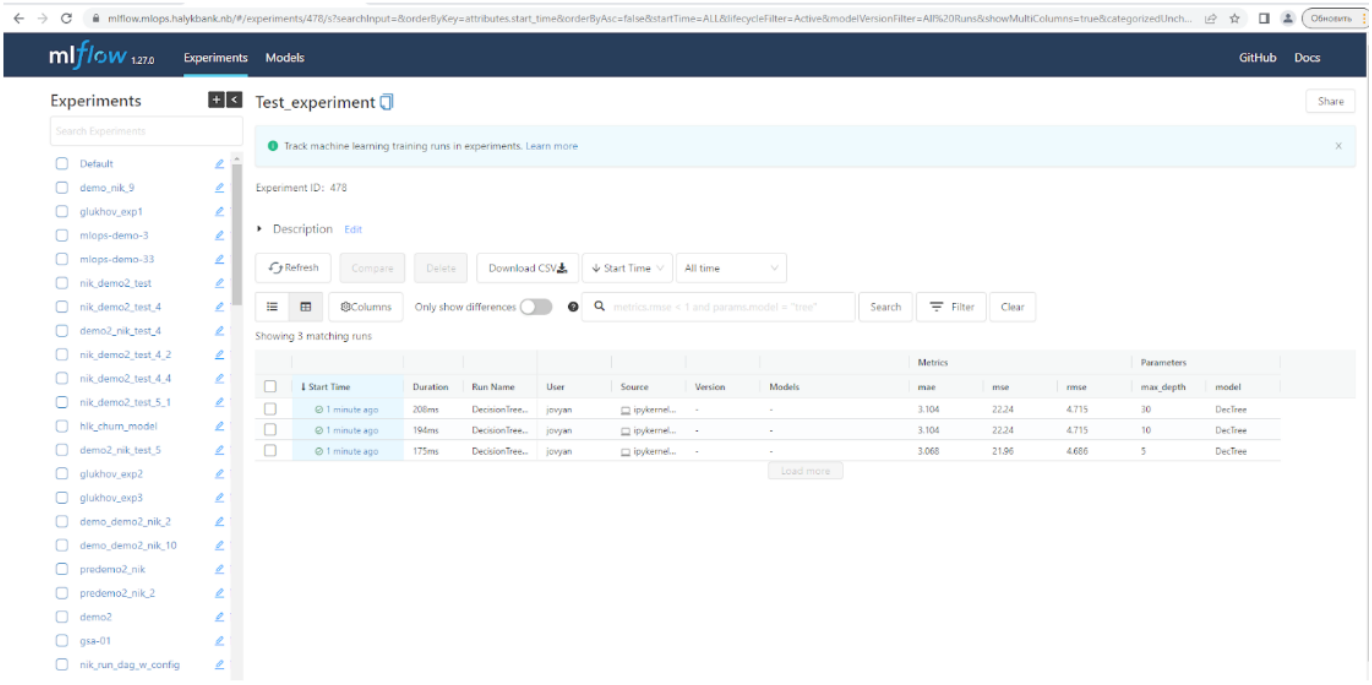
На 21 строке выполняется предсказание (скоринг).

На строках 22–24 происходит расчет метрик обученной модели.

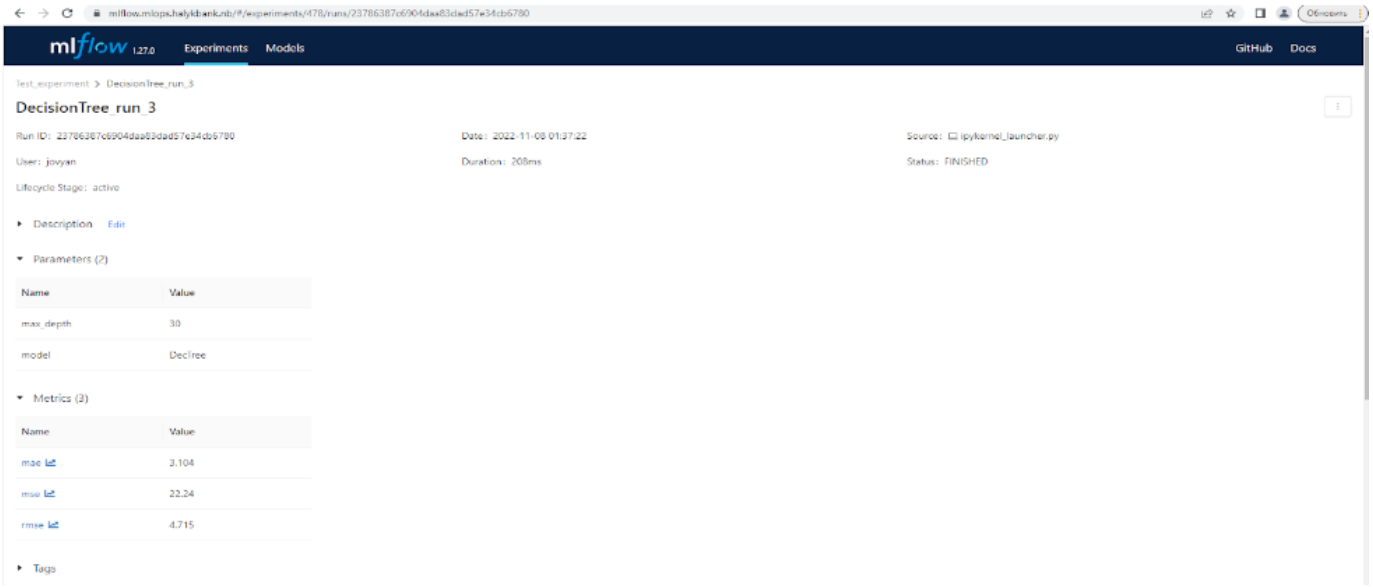
На строках 26–28 происходит логирование рассчитанных выше метрик модели, название метрики указывается первым аргументом как строка, а значение метрики указывается вторым аргументом.

На строках 29–30 выполняется логирование параметров модели, где первым параметром передается строка, задающая имя параметра, а вторым - значение данного параметра.

Таким образом, после запуска данного кода в реестре экспериментов MLflow будет создан эксперимент с именем Test_experiment, который содержит три запуска (три рана) с названиями (см. рис. ниже). При этом будут указаны метрики mae, mse, rmse для трех вариантов решающего дерева, отличающихся параметром максимальной глубины 5, 10 и 30.



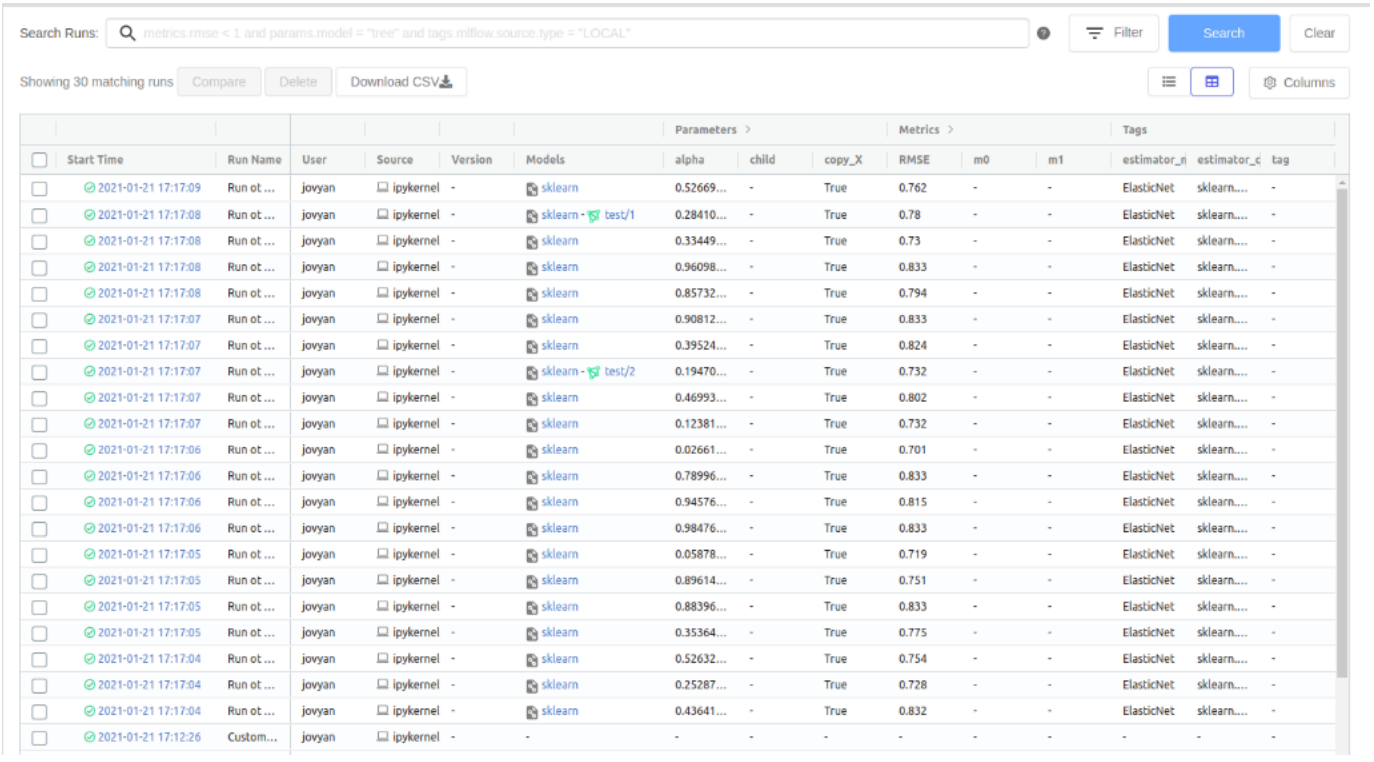
Если открыть форму запуска «run» (“провалиться” внутрь) в какой-либо запуск (ран), то на экране будут отображены логированные атрибуты модели, в данном примере это два параметра и три метрики (см. рис. ниже).



Также бывает полезным логировать тэги и саму модель, более подробно про возможности логирования можно прочитать в документации Mlflow https://www.mlflow.org/docs/latest/index.html.

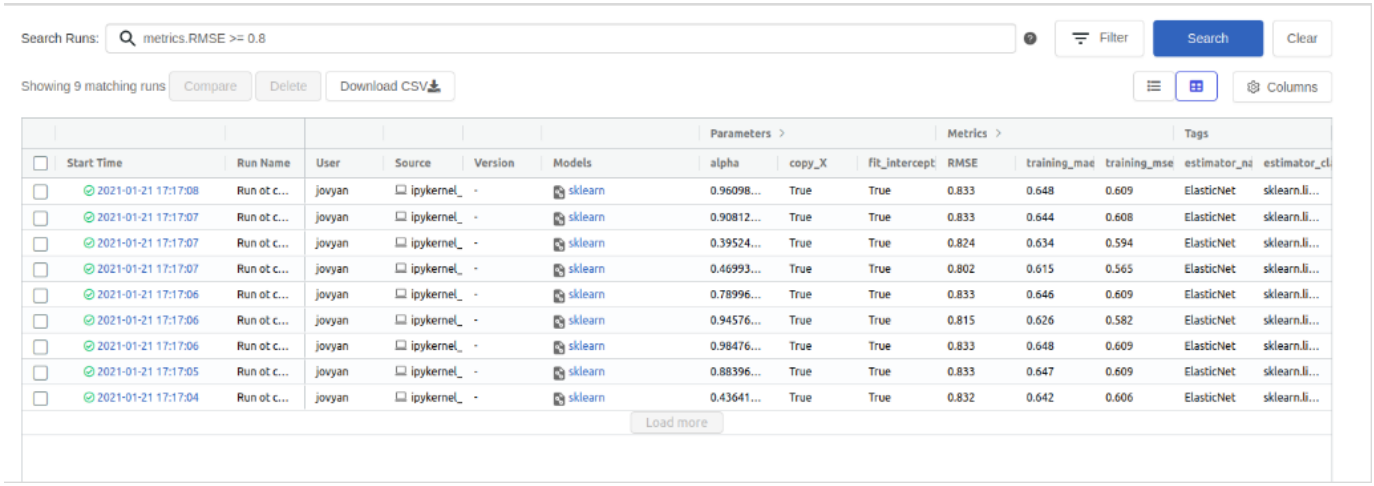
Поиск запуска по параметрам

В случае, когда эксперимент содержит большое количество runs, то оказывается удобным воспользоваться фильтрацией отображаемые runs по параметру. Рассмотрим пример, перейдя на основную страницу MLflow.



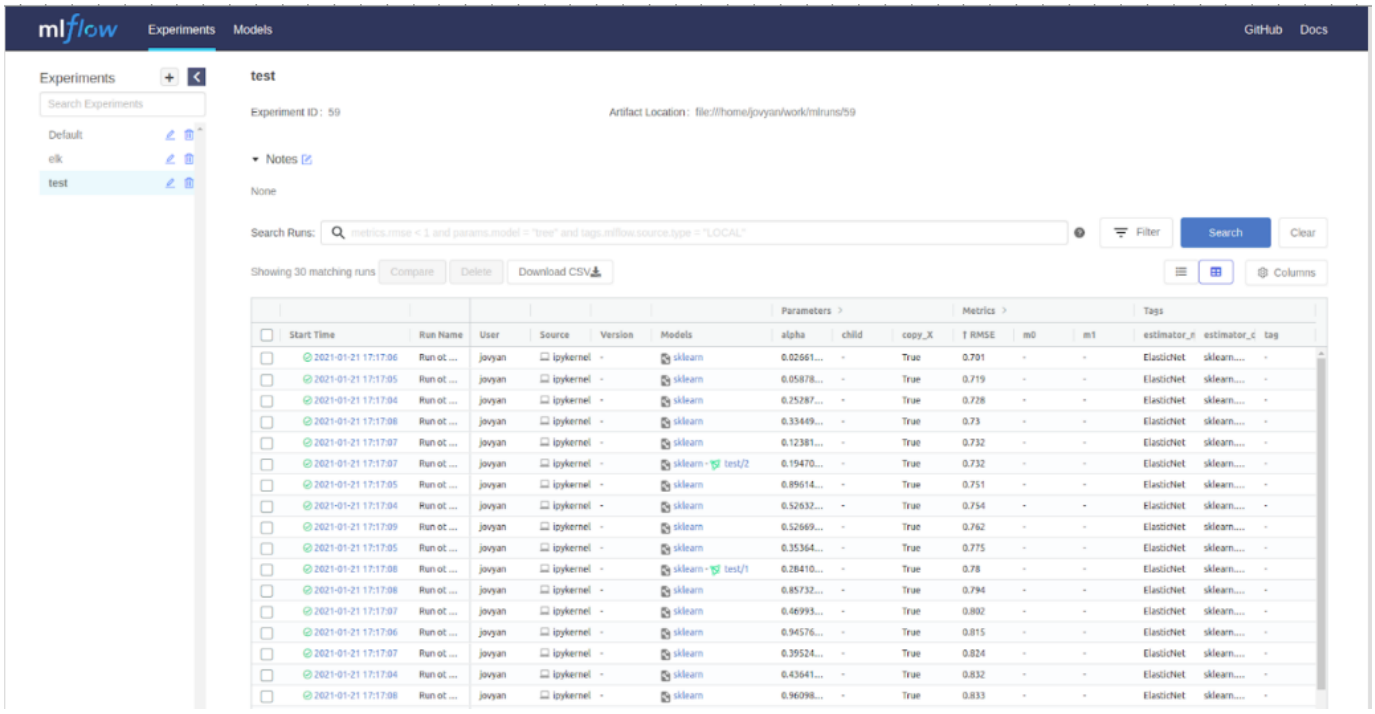
На странице списка запусков (run) в поле поиска ввести строку с параметрами, например: metrics.RMSE >= 0.8

И нажать на кнопку **“Search”** для поиска запусков, но на экране отобразятся только запуски, удовлетворяющие данному критерию.



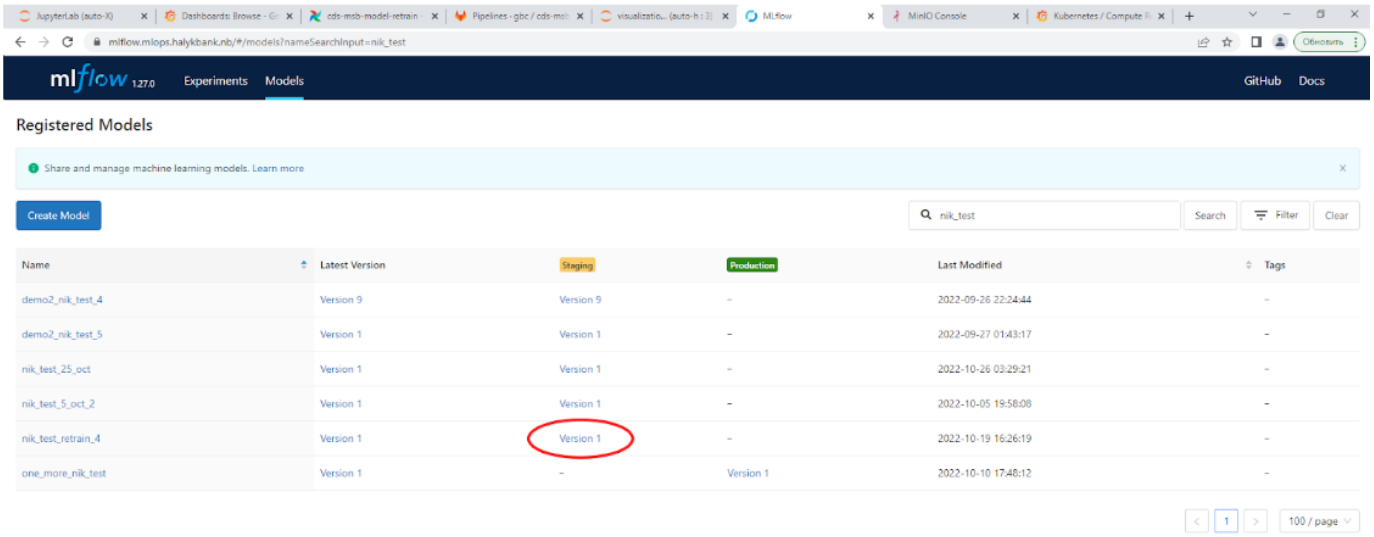
Сортировка по значению в столбце

Запуски можно отсортировать по значениям в порядке убывания (возрастания) по одному столбцу, нажав на соответствующий столбец.

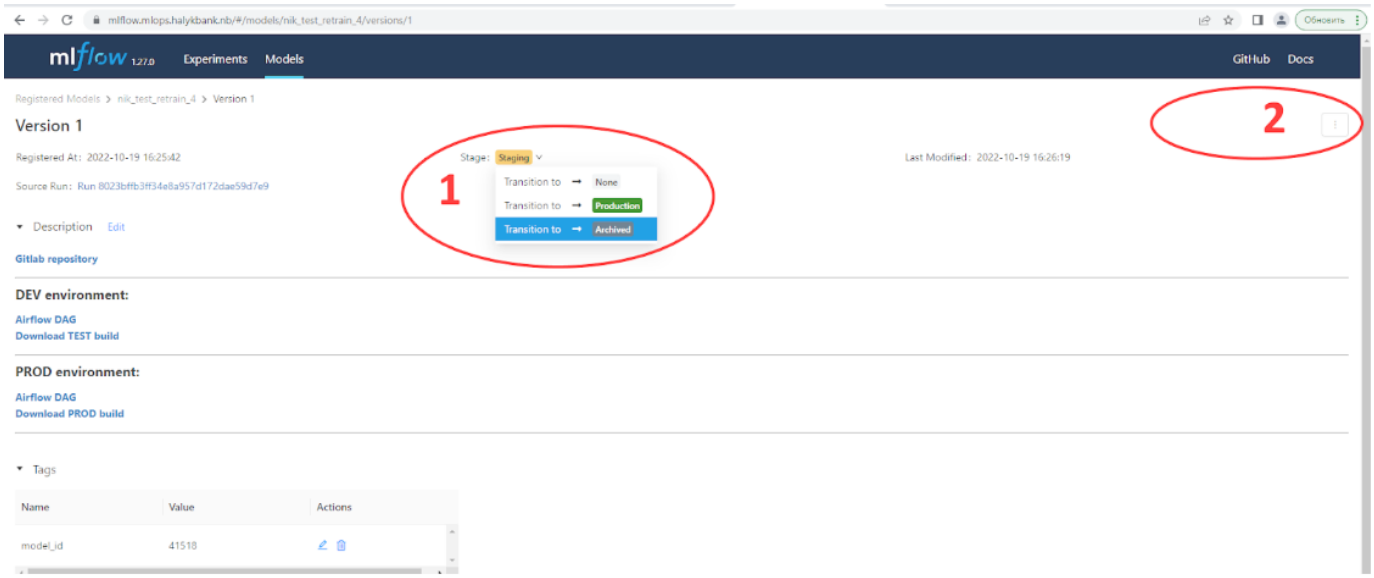


Удаление модели и эксперимента

Для удаления модели из реестра моделей MLflow сперва требуется перевести модель в стадию “Archived”. Для этого нужно в реестре моделей выбрать версию, находящуюся на самой высокой стадии, в данном примере на рисунке ниже это стадия “Staging”.

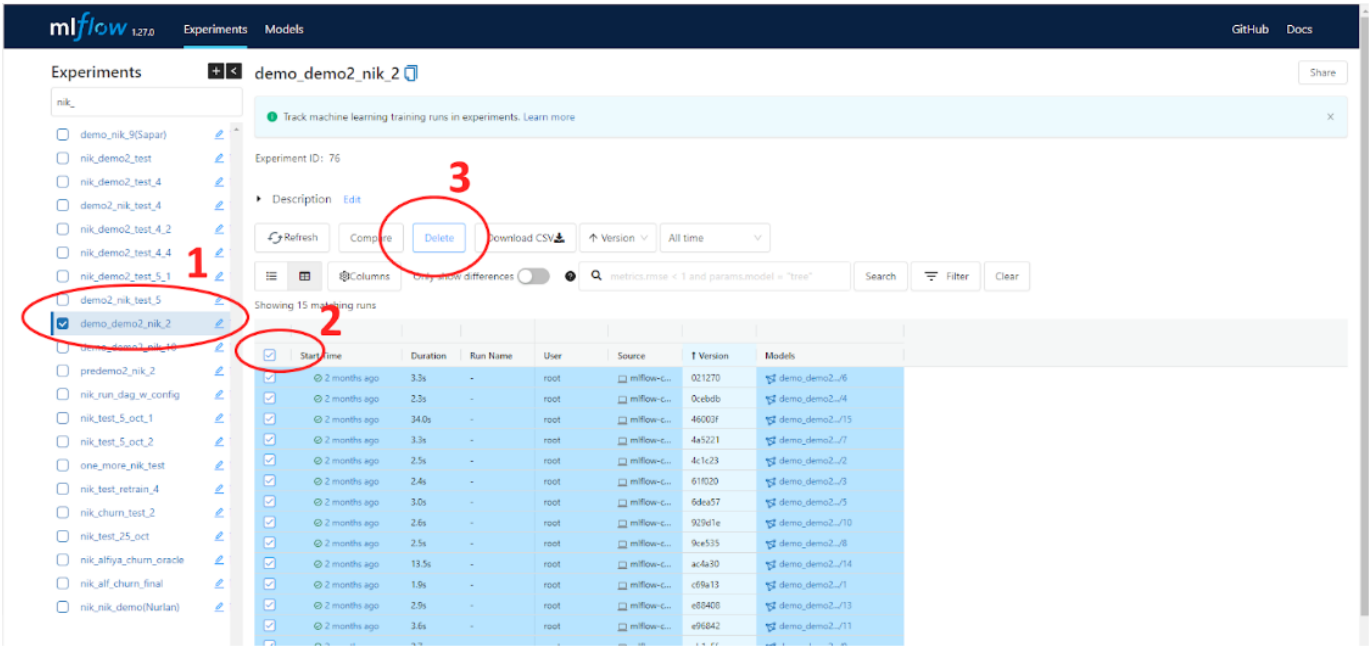


Для перевода в стадию “Archived” нужно нажать на поле “Stage”, выбрать “Transition to Archived”. После этого в меню под цифрой 2 на рисунке ниже появиться возможность удалить (Delete), после нажатия на Delete, появиться меню с предупреждением, что действие безвозвратно удалит версию модели со всеми артефактами, нужно ввести подтверждение.



После того как все версии модели удалены, тогда при нажатии на название модели в реестре, будет открыто окно, подобное тому, как на рисунке выше, в этом окне нужно выбрать значок с 3 точками (цифра 2 на рисунке) и еще раз прожать удаление модели. После этого модель будет удалена из реестра моделей.

Полное удаления эксперимента из таблицы “Experiments” в силу особенностей архитектуры MLflow проблематично, однако для нормальной работы на Платформе достаточно удалить все run's внутри данного эксперимента. Рассмотрим процесс на рисунке ниже.



Сначала выбираем имя эксперимента (цифра 1), все запуски которого требуется удалить. Далее выбираем все или некоторые запуски (цифра 2), далее нажимаем кнопку “Delete” (цифра 3), после этого еще раз подтверждаем удаление. Теперь выбранные запуски будут удалены.

Интеграция с DVC и S3

ОБЩАЯ ИНФОРМАЦИЯ

DVC представляет собой инструмент для версионирования нетекстовых данных (например, файл с моделью *.pkl или датасеты в* .pkl), дополняющий возможности Git. Также он помогает обеспечивать полноценную воспроизводимость экспериментов машинного обучения.

DVC связывает код модели и версии данных, это в свою очередь предоставляет возможность версионировать данные всевозможных типов и размеров, использовать удаленное хранилище для хранения и передачи, а также позволяет сохранять и сравнивать различные метрики моделей.

Благодаря DVC существует возможность в Git хранить данные в виде метаданных, а сами данные располагать в удаленном или локальном хранилище в другом месте.

DVC является своего рода плагином для S3 хранилища. Основным понятием в S3 является бакет (“bucket”-корзина), который можно сравнить с директорией в файловой системе. На платформе по умолчанию реализовано два бакета: первый бакет используется для хранения файлов атрибутов моделей, а второй выступает в роли хранилища для информации, которая логируется в инструменте Mlflow.

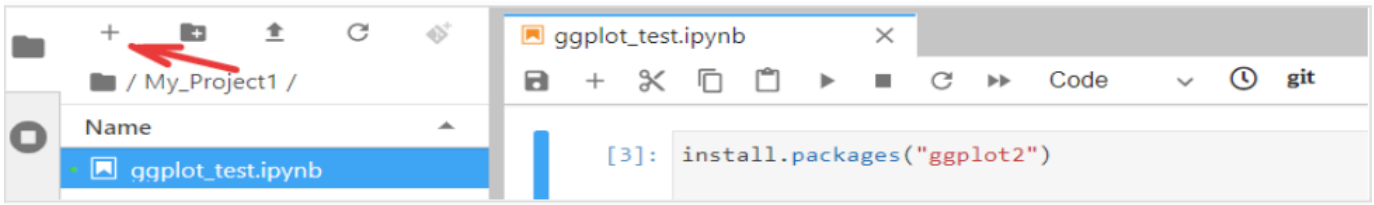
РАБОТА С ТЕРМИНАЛОМ В JUPYTERLAB

Терминал JupyterLab предоставляет возможность работы с интерпретаторами команд операционной системы, такими как bash, tcsh и т. д.

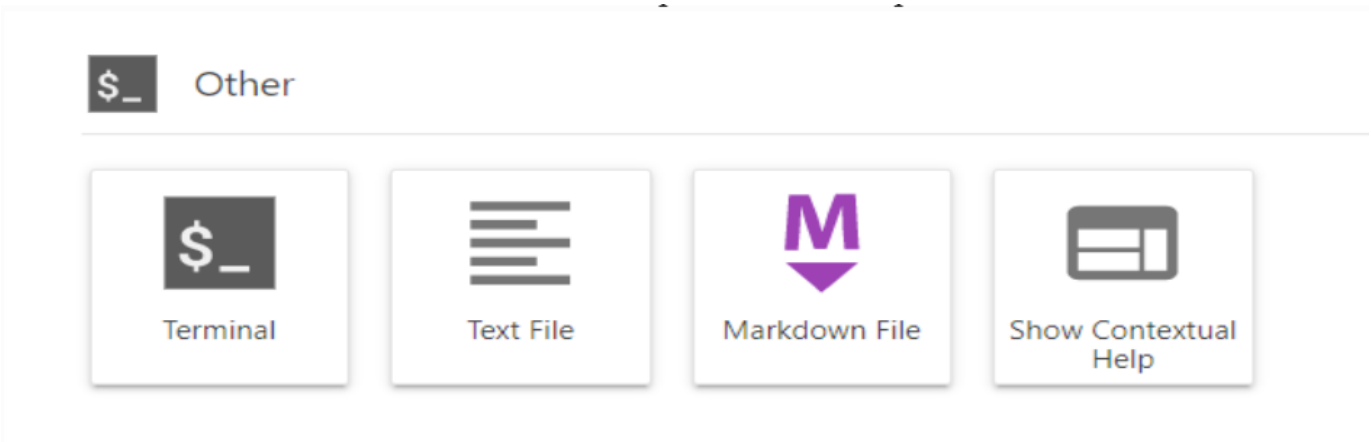
Терминалы запускаются в рамках того же инстанса, на котором запущен ваш Jupyter Server. Команды выполняются с правами вашего пользователя.

Работа с терминалом может быть полезна или даже необходима, в случаях, когда требуется установить новые библиотеки/фреймворки, при работе с системой контроля версий Git или для сбора информации при возникновении ошибок.

Для того, чтобы открыть новый терминал нажмите на символ “+” в левой панели основного интерфейса.



В появившейся вкладке **“Launcher”** выберите новый Терминал.



Примечание: подробно о работе с Терминалом Jupyter Lab можно прочитать в официальной документации: https://jupyterlab.readthedocs.io/en/stable/user/terminal.html

ОСНОВНЫЕ КОМАНДЫ

Интеграция Git с DVC происходит на этапе работы с pkl-файлами и файлами, имеющими большой объем (архивы, датасеты, картинки). Система контроля версий Git не предназначена для хранения и версионирования бинарных (и других нетекстовых) файлов, особенно когда эти файлы большого объёма. В развернутой Системе подразумевается, что в DVC будут отслеживаться *.pkl файлы с обученной моделью (например, мы обучили модель sklearn.tree.DecisionTreeClassifier и сохранили этот объект в файл model.pkl). Опционально, можно сохранить в DVC датасеты, на которых модель обучалась, в этом случае нужно убедиться, что в удаленном S3 хранилище достаточно свободного места для записи, так как объем датасетов может быть достаточно большим.

Чтобы сохранить в хранилище S3 сериализованный файл с моделью model.pkl, который должен находиться в директории pkl, нужно выполнить следующую команду в терминале (текущей директорией терминала должен быть корень текущего проекта):

```
dvc add ./pkl/model.pkl
```

После выполнения команды в папке pkl помимо model.pkl появится файл model.pkl.dvc, в котором указан хэш файла, размер и название, по которым DVC сможет определить с каким файлом model.pkl связан наш коммит в Git.

Затем следует выполнить команду:

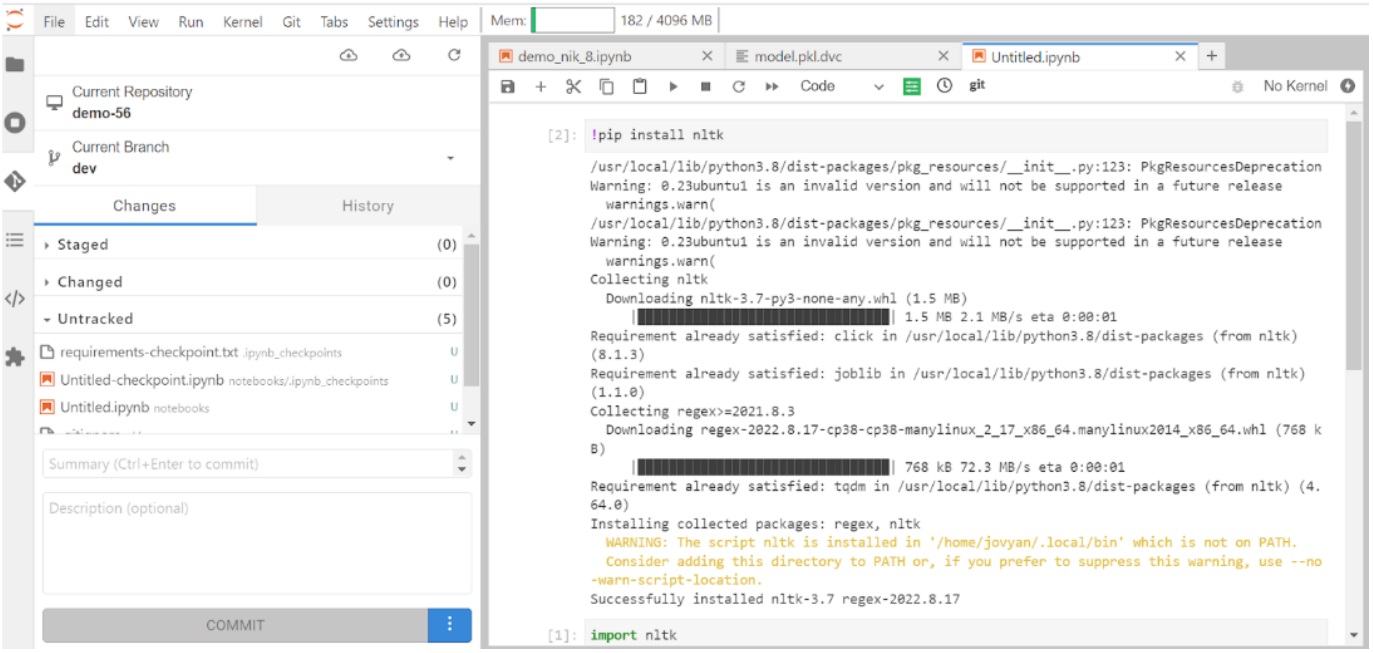
```
dvc push
```

Внимание! Для дальнейшего удобства работы в системе следует придерживаться подхода, при котором для каждой версии модели сначала выполняются команды `dvc add`, `dvc push`, а затем `git add -> git commit -> git push origin dev`. При таком подходе будет легко ориентироваться в коммитах git, поскольку каждому из них будет соответствовать версия модели в DVC.

После выполнения команды файл model.pkl, будет сохранен в удаленном S3 хранилище, а Пользователь при коммите в git будет должен добавить на отслеживание (track) файл в директории pkl/model.pkl.dvc (подробнее см. п. «Взаимодействие с системой контроля версий Git»). Отметим, что в этом файле содержится md5 хэш, причем первые два символа в нем используются как имя для папки в бакете S3, в которую непосредственно отправляет сам model.pkl на хранение. Например, если в файле model.pkl.dvc строка с md5 хэшем выглядит так “md5: df175d022e2e43b425d1ea70adc439f6”, то значит файл model.pkl в S3 внутри бакета ‘b-a’ будет создана папка с названием “df”, в которую и будет загружен файл с моделью.

Взаимодействие с системой контроля версий Git

Для переноса файлов в Git необходимо в боковом меню нажать на значок Git.



После этого необходимо выполнить следующие действия:

1. Раскрыть Untracked (список не отслеживаемых в Git файлов);
2. Для каждого необходимого файла (как правило, это все .py файлы, файл requirements.txt, файлы jupyter ноутбуков, и обязательно файл model.pkl.dvc из папки pkl) для переноса файлы, нажав на плюсик справа от названия файла, после чего они переместятся в раздел Staged;
3. Написать текст коммита в окне Summary;
4. Нажать на кнопку “Commit”, после чего появится упоминание о commit в разделе History.

Далее для отправки в Git необходимо нажать на кнопку, расположенную в верхней части экрана, после чего файлы появится в Git репозитории в ветке dev.

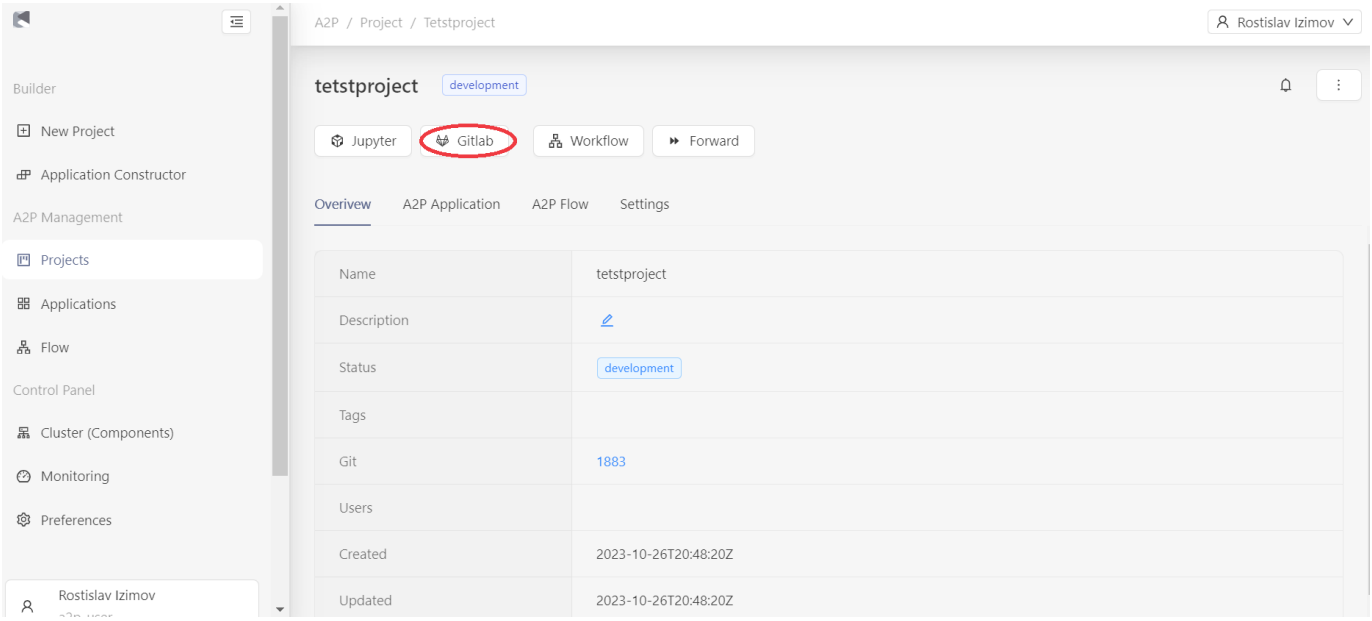
Работа с GitLab

ПОДКЛЮЧЕНИЕ К WEB-ИНТЕРФЕЙСУ

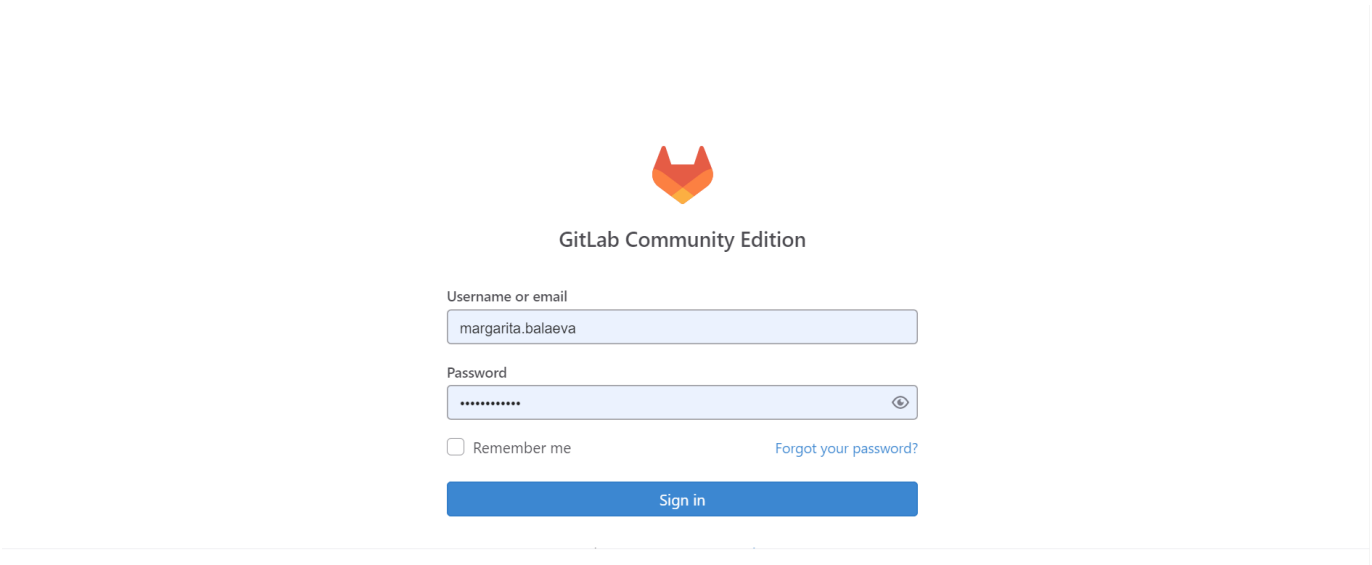
Для того, чтобы обеспечить эффективную совместную работу пользователей, а также для автоматизации CI/CD процессов используется система контроля версий Git (Gitlab SCM).

В системе контроля версий отслеживаются файлы блокнотов (ipynb), Python-файлы, конфигурационные файлы и файлы зависимостей (например, requirements.txt).

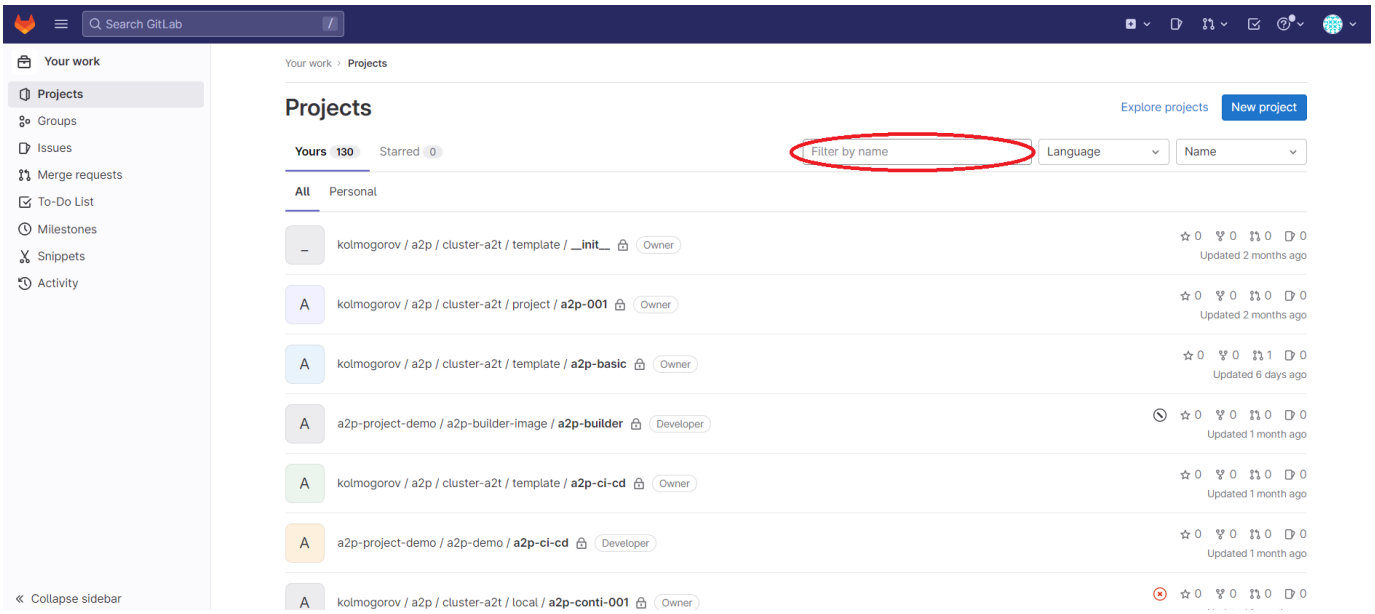
Для входа в web-интерфейс из модуля A2P требуется перейти по "GitLab" (см. рисунок ниже):



Далее ввести данные вашей учетной записи (рисунок ниже):



После входа в web-интерфейс будет отображена панель со всеми имеющимся репозиториями Пользователя, после этого нужно найти свой репозиторий своего проекта и перейти в него, если проектов много, то удобно воспользоваться окном поиска (см. рисунок ниже).



ОБЗОР ВЕТОК РЕПОЗИТОРИЯ В GITLAB

Следует напомнить, что репозиторий в GitLab, имеющий имя проекта разрабатываемой модели создается при запуске create_project.ipynb еще на этапе разработки модели в Jupyter.

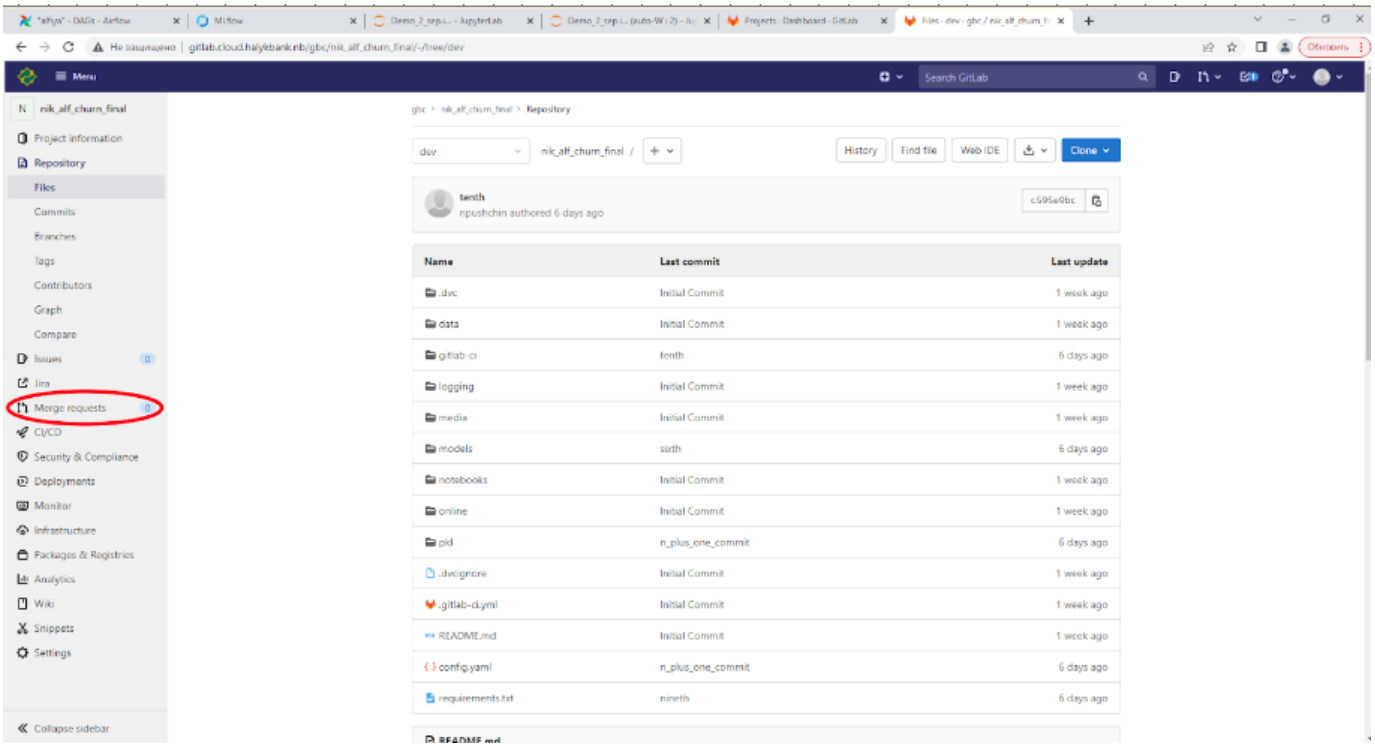
В созданном репозитории имеются необходимые конфигурационные файлы и следующие ветки:

- dev - ветка разработки модели;
- main - ветка, при успешном коммите в которую запускается CI/CD процесс в среде DEV;
- prod - ветка, при успешном коммите в которую запускается CI/CD процесс в промышленной среде (PROD) для Batch моделей;
- retrain - ветка, коммит в которую запускает процесс переобучения модели;
- online - ветка, при успешном коммите в которую, запускается CI/CD процесс в промышленной среде (PROD) для Online моделей.

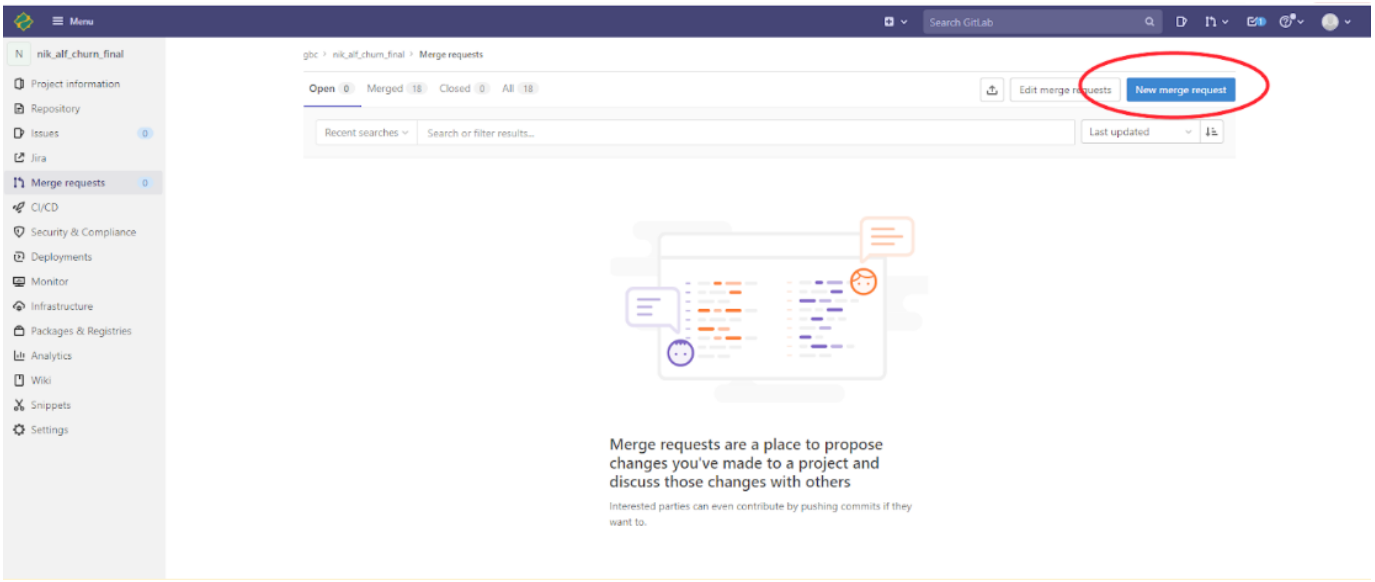
CI/CD ПРОЦЕСС ДЛЯ ПРОДУКТИВИЗАЦИИ МОДЕЛИ В СРЕДЕ DEV

После того как Пользователь запушил коммит из контейнера разработки модели (контейнер, в котором велась работа в Jupyter), необходимо перейти в удаленный репозиторий в GitLab в ветку dev и проверить наличие данного коммита.

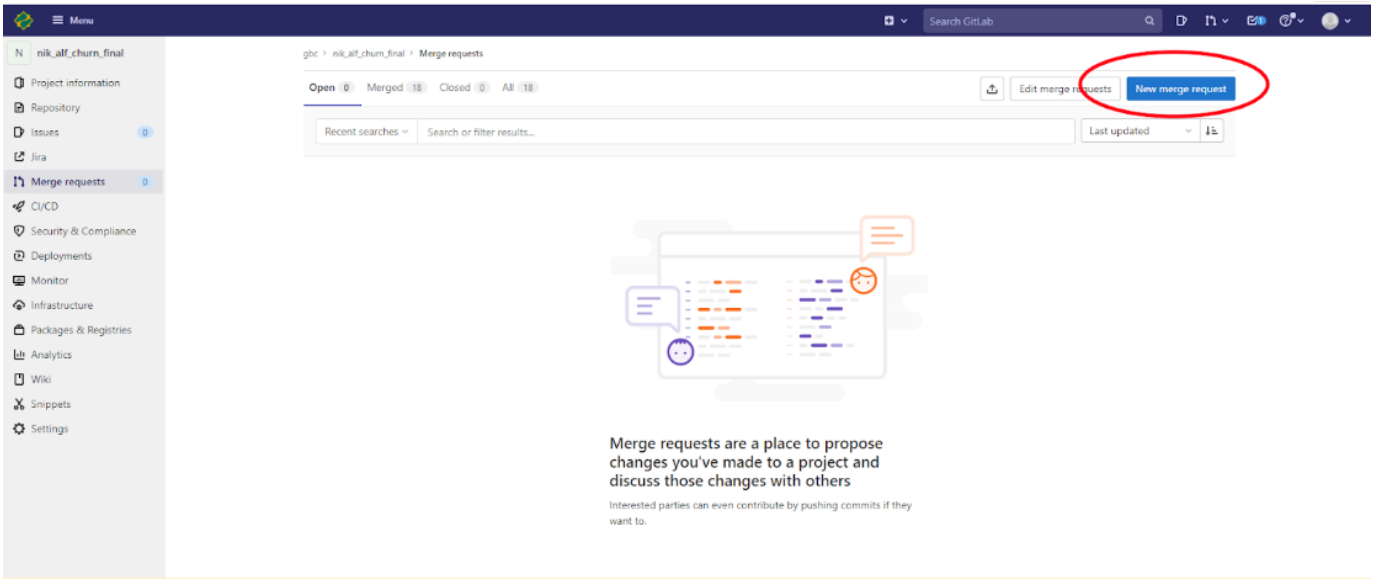
Далее, запуск автоматического CI/CD процесса продуктивизации модели в среде DEV инициируется с любого коммита в ветку main, для этого требуется нажать на кнопку «Merge Request» из панели пользователя, рисунок ниже.



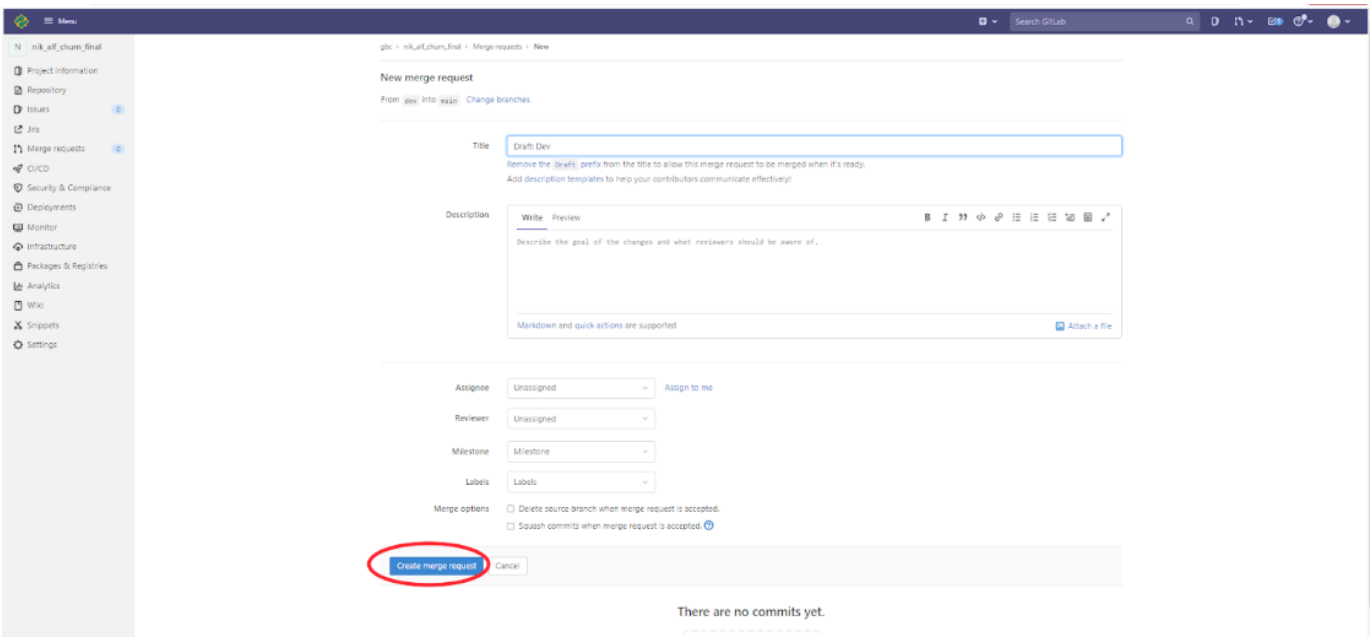
Далее нажать на “New merge request” (рисунок ниже):



Далее в поле Source branch необходимо выбрать dev, а в поле Target branch выбрать main и нажать “Compare branches and continue” (см. рис. ниже):

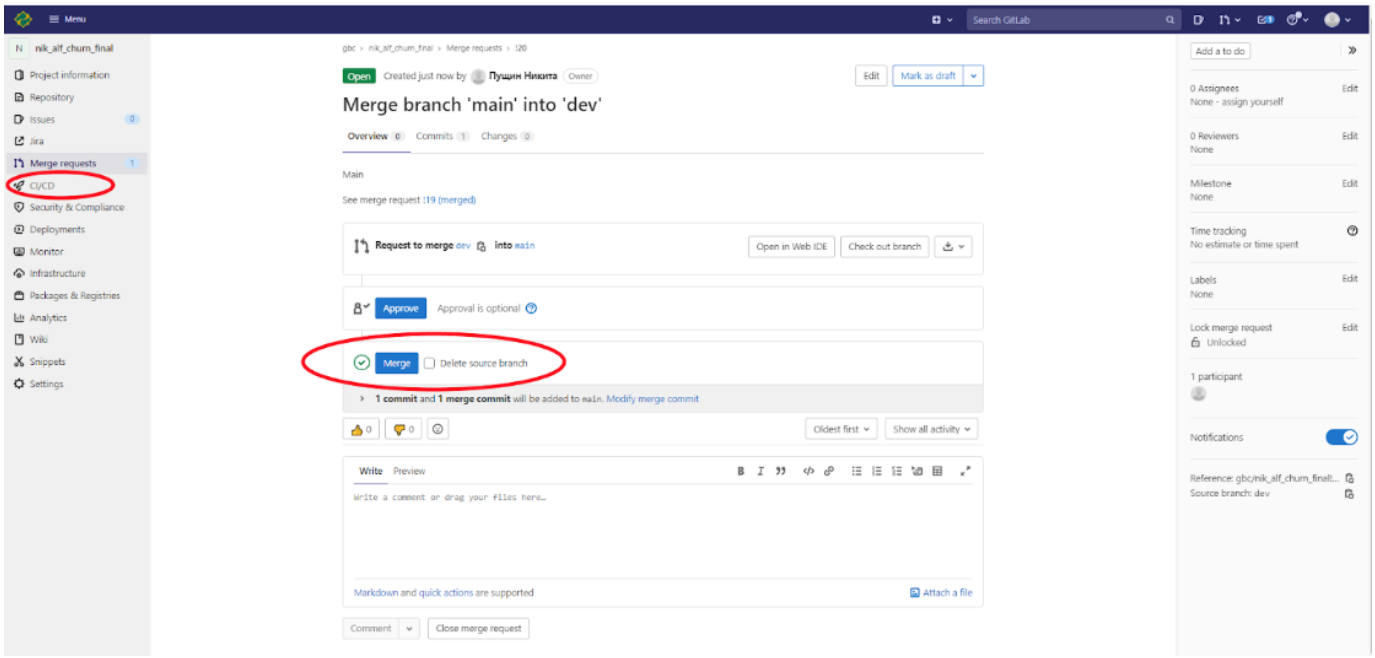


Далее, если необходимо, то заполнить поля Description, Assignee, Reviewer, Milestone, Labels и нажать “Create merge request” (рисунок ниже и описание Description, Assignee, Reviewer, Milestone, Labels указано ниже).

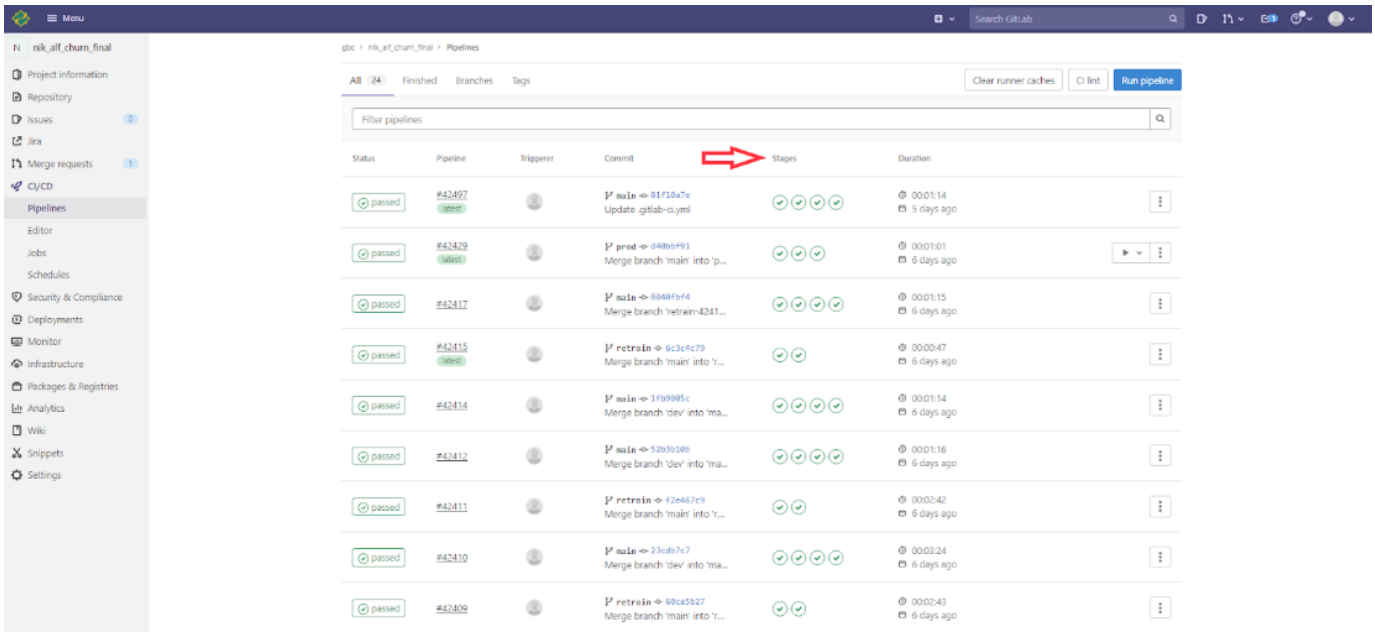


- Description – Описание Пользователем разработанной модели;
- Assignee – Ответственный за запрос;
- Reviewer – Рецензент запроса;
- Milestone – Временные рамки, специально установленные Пользователем, для конкретных целей;
- Labels – Указываются runner или группы runners, которые используются для различных задач.

На открывшейся странице нужно нажать на кнопку “Merge” и перейти на вкладку CI/CD слева (рисунок ниже):



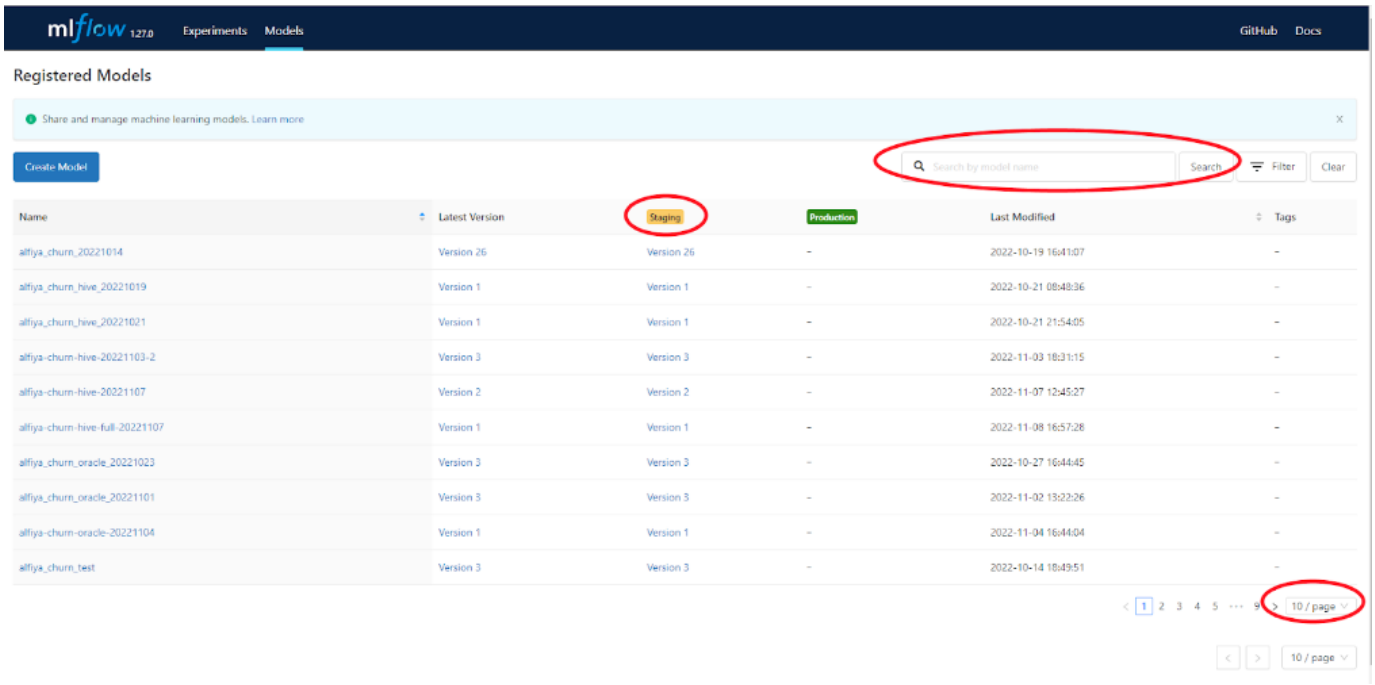
На рисунке ниже представлено страница с CI/CD пайплайнами для текущей модели.



Пайплайн продуктивизации модели в среде DEV состоит из 4 стадий (stages):

- build (сборка образа с моделью);
- create-model (создание модели в реестре моделей в MLflow);
- deploy (создание DAG-файла и загрузка в репозиторий airflow-dev);
- stage-model (переводит модель из стадии Latest в стадию Staging, см. рисунок ниже).

Таким образом, при успешном прохождение всех 4 стадий процесса можно перейти в MLflow и на вкладке Models увидеть последнюю версию и стадию своей модели. Если модель не найдена на экране, то можно воспользоваться поиском; еще полезно увеличить число отображаемых на одной странице моделей справа внизу, рисунок (MLFlow модель).

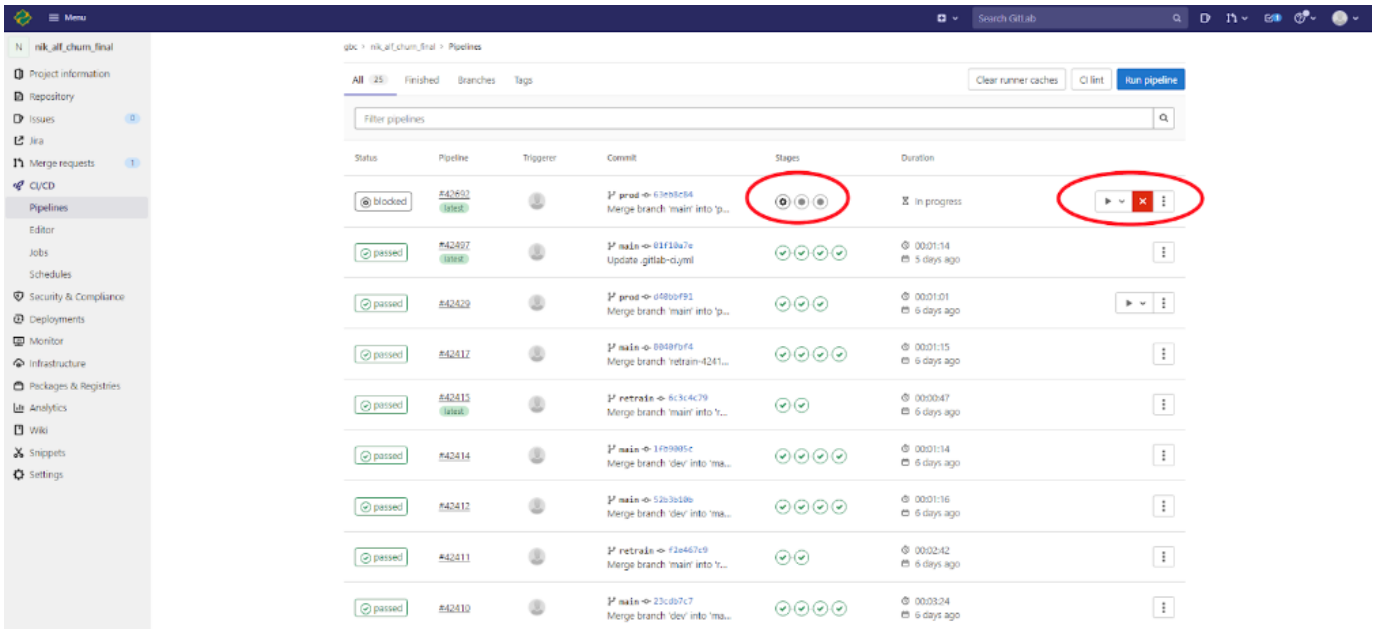


После окончания CI/CD процесса можно переходить в Airflow, где должен появиться DAG файл с названием проекта. Время появления DAG-файла в Airflow от окончания CI/CD процесса может занимать до нескольких минут. Далее можно активировать DAG-файл, более подробно про активацию DAG-файла в Airflow написано в разделе «Оркестрация рабочих процессов машинного обучения».

CI/CD ПРОЦЕСС ДЛЯ ПРОДУКТИВИЗАЦИИ МОДЕЛИ В СРЕДЕ PROD

После того, как проверена работа модели в контуре DEV, она может быть переведена в контур PROD, для этого требуется выполнить Merge Request по аналогии с CI/CD процессом продуктивизации модели в контуре DEV, только в качестве Source Branch должна быть main, а в качестве Target Branch - prod. После успешного выполнения запроса слияния следует перейти в меню слева на вкладку CI/CD, на экране будет пайплайн, состоящий из 3 стадий. На первой стадии выполняется сборка docker образа с моделью, на второй стадии создается DAG-файл для модели и размещается в репозитории airflow-prod, на третьей стадии в реестре моделей MLflow производится перевод данной модели из стадии Staging на стадию Production.

При выводе в PROD каждая стадия процесса требует ручного запуска (значок шестеренка), для этого в колонке справа следует нажать на запуск (рисунок ниже).



После успешного окончания CI/CD процесса продуктивизации модели в контуре PROD можно переходить в Airflow-prod и выполнить установку DAG-файла на запуск по регламенту (расписанию), более подробно про работу в Airflow можно прочитать в разделе «Оркестрация рабочих процессов машинного обучения». Обратите внимание, что, в соответствии с ролевой моделью (см. Приложение А. «Ролевая Модель»), не все пользователи имеют возможность активации DAG-файлов модели в контуре PROD.

CI/CD ПРОЦЕСС ДЛЯ ПЕРЕОБУЧЕНИЯ МОДЕЛЕЙ

В случае, если модель подразумевает переобучение, то на этапе разработки модели должен быть подготовлен файл retarin.py и dag-retrain.py. Логика процесса переобучения построена следующим образом: необходимо сделать коммит в ветку retrain, как правило, таким коммитом является merge request из ветки main в ветку retrain. Коммит автоматически запустит CI/CD процесс для переобучения модели, который состоит из двух стадий. На первой стадии формируется docker образ для применения кода переобучения (упрощенно, это контейнер, в котором будет запущен скрипт retrain.py). На второй стадии формируется DAG-файл для переобучения, имеющий имя проекта модели с приставкой “retrain” в конце, данный файл загружается в репозиторий airflow-dev. Далее, спустя обычно несколько минут, данный файл становится виден в интерфейсе Airflow, где он доступен для активации. Нужно его там активировать (более подробно про работу с Airflow смотрите в разделе «Оркестрация рабочих процессов машинного обучения»), после активации DAG-файл должен встать на расписание. Далее, по наступлению условия активации, данный DAG-файл должен отработать и, в случае отсутствия ошибок, в репозитории с моделью должна появиться новая ветка с названием типа retrain-pipeline_id-date, где data – это дата запуска DAG-файла переобучения, а pipeline_id – это номер CI/CD пайплайна. В этой ветке в папке pkl будет находиться свежеобученная модель в файле model.pkl.dvc. Далее нужно выполнить merge request из данной ветки retrain-pipeline_id-date в ветку main (для продуктивизации в среде dev) или в ветку prod (для продуктивизации в среде PROD). После слияния (“merge”) автоматически запустится CI/CD продуктивизации модели для среды dev (подробнее см. п. «CI/CD процесс для продуктивизации модели в среде DEV») или prod (подробнее см. п. «CI/CD процесс для продуктивизации модели в среде PROD»).

Важно отметить, если переобучаемая модель стояла на расписании в Airflow и, например, проводила скоринг ежедневно, то после окончания CI/CD процесса, о котором шла речь выше (после мерджа retrain-pipeline_id-data в main или prod), будет автоматически будет обновлен DAG-файл с помощью, которого происходит скоринг моделью. То есть, на следующий день данная модель будет делать предсказания с помощью свежеобученной модели.

Таким образом, порядок действий Пользователя при переобучении, следующий:

1. Сделать merge request из ветки main в ветку retrain.
2. Перейти в Airflow и активировать DAG-файл переобучения.
3. В репозитории модели должна появиться новая ветка retrain-pipeline_id-date, где в папке pkl должен быть файл model.pkl.dvc, который и является свежеобученной моделью.
4. Выполнить merge request из ветки retrain-pipeline_id-date в ветку main (для переобучения модели, продуктивизированной в среде dev) или prod (для переобучения модели, продуктивизированной в среде prod).
5. После этого в Airflow соответствующей среды DAG-файл модели будет автоматически обновлен и при его следующем запуске, скоринг будет проводиться уже с помощью свежеобученной модели.

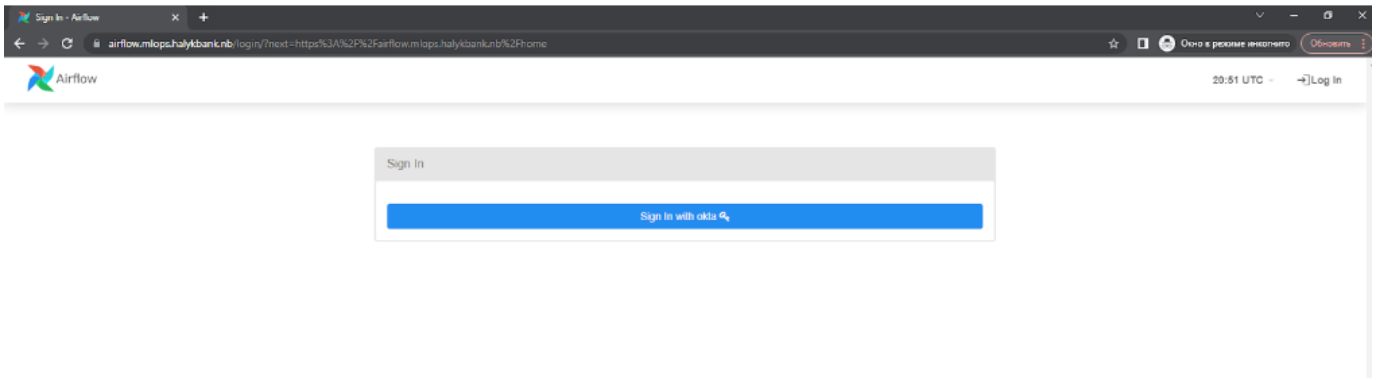
Оркестрация рабочих процессов машинного обучения

Airflow является оркестратором процессов запуска моделей. Именно Airflow ответственен за запуск скоринга моделей и их переобучения по расписанию. Основным объектом в Airflow является DAG-файл (DAG- directed acyclic graph, то есть прямой ациклический граф).

ИСПОЛЬЗОВАНИЕ ВЕБ-ИНТЕРФЕЙСА AIRFLOW

В Системе реализовано два инстанса (экземпляра) Airflow: Airflow – для отладки моделей в среде DEV и Airflow-prod – для моделей в контуре PROD.

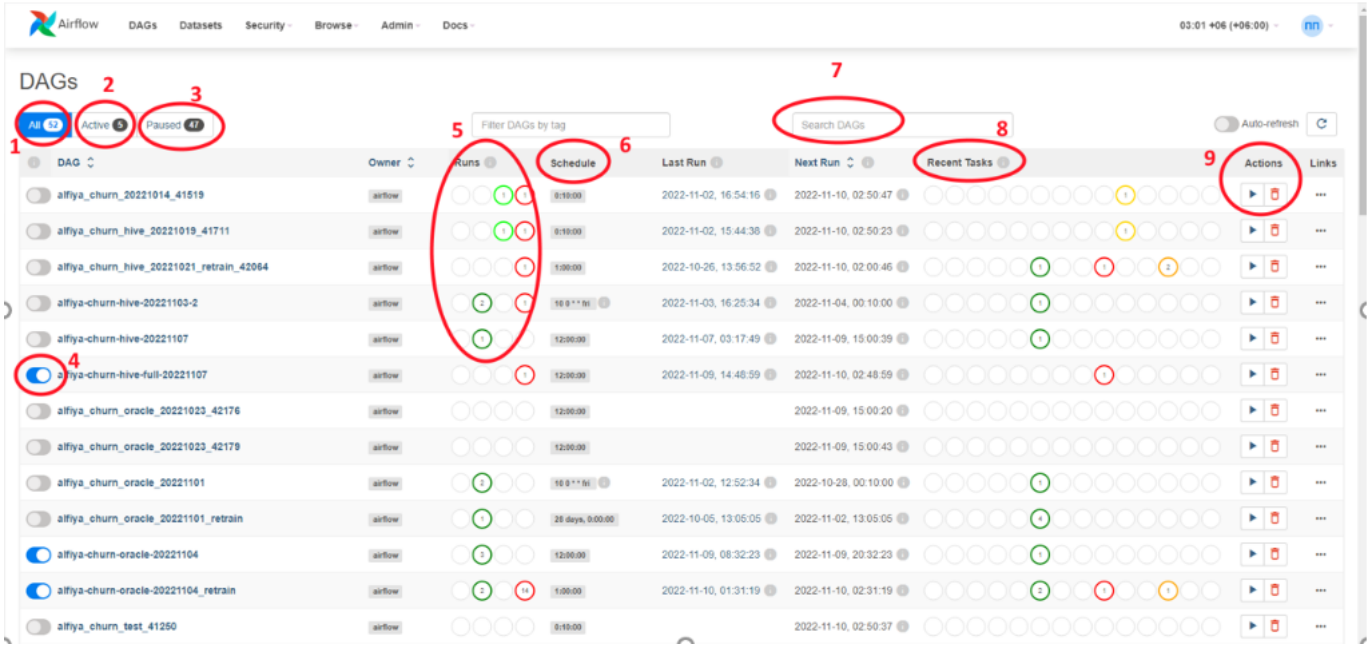
При первом переходе по данным адресам в браузере откроется страница (рисунок ниже), где нужно нажать на “Sign in with okta”, после этого откроется страница авторизации keycloak, где нужно будет ввести свои учетные данные.



После входа в Airflow (не важно в dev или prod) на экране будет основная страница интерфейса.

Главная страница Airflow представляет собой таблицу, в которой отображаются DAG-файлы. Для Airflow в среде DEV данные DAG-файлы берутся из репозитория airflow-dev (в группе gbc) в GitLab. А в этот репозиторий они попадают на третьей стадии CI/CD процесса, который инициируется любым коммитом в ветку main репозитория с моделью. Название DAG-файла для скоринга (то есть запуска скрипта main.py внутри контейнера) совпадает с именем проекта модели, а название DAG-файла для переобучения модели состоит из названия проекта модели и далее слова retrain. Отображаемые в интерфейсе DAG-файлы переобучения также берутся из репозитория airflow-dev, в который они попадают на второй стадии CI/CD процесса переобучения модели, который инициируется любым коммитом в ветку retrain.

Рассмотрим основные поля главной страницы (рисунок ниже):



По умолчанию, вновь появившийся (после соответствующего CI/CD процесса) DAG-файл попадает в таблицу и является неактивным, другое название “стоящим на паузе” (серый ползунок в столбце слева). Если DAG-файл не активен, то он не будет запущен по расписанию. Если перевести этот ползунок в состояние активации или, “затриггерить” (как, например, на цифре 4 рисунка), то теперь Airflow будет отслеживать расписание запуска, заданное внутри данного файла и, если наступит условие запуска, то запустит DAG-файл.

Под цифрами 1, 2, 3 обозначены фильтры, показывающие все файлы (1), показывающие только активные файлы (2) и не активированные файлы (3).

В столбце под цифрой 6 указано расписание запуска данного файла, можно навести на это поле мышкой и увидеть более подробную информацию про расписание.

Для поиска нужного DAG-файла можно воспользоваться поиском, цифра 7.

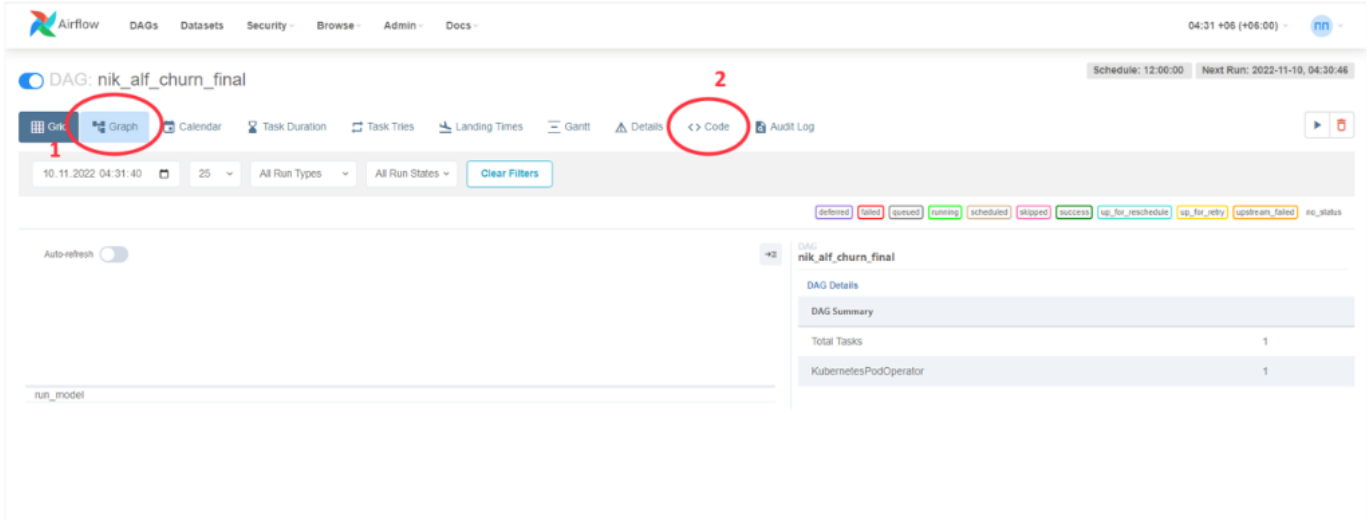
Под цифрой 5 указано число запусков (runs) данного файла, а цветом указан статус запуска:

- Салатовый означает что запуск идет прямо сейчас,
- Темно-зеленый означает, что запуск прошел успешно,
- Красный - запуск окончился ошибкой,
- Светло-серый - запуск поставлен в очередь.

Под цифрой 8 указаны последние tasks (таски) запуска. В нашем случае, для скоринга модели таска тождественна рану, поскольку ран содержит одну таску. А в случае с переобучением один ран содержит четыре таски. Для упрощения можно считать, что таска – это стадия работы DAG-файла, а если более точно, то каждая таска описывается как объект KubernetesPodOperator внутри DAG-файла. Напомним, что в разделе “Описание файла dag-retrain.py” в примере кода имеется четыре объекта KubernetesPodOperator, а в файле “dag-batch.py” описан один KubernetesPodOperator, поэтому DAG-файл скоринга содержит одну таску, а DAG-файл переобучения - четыре.

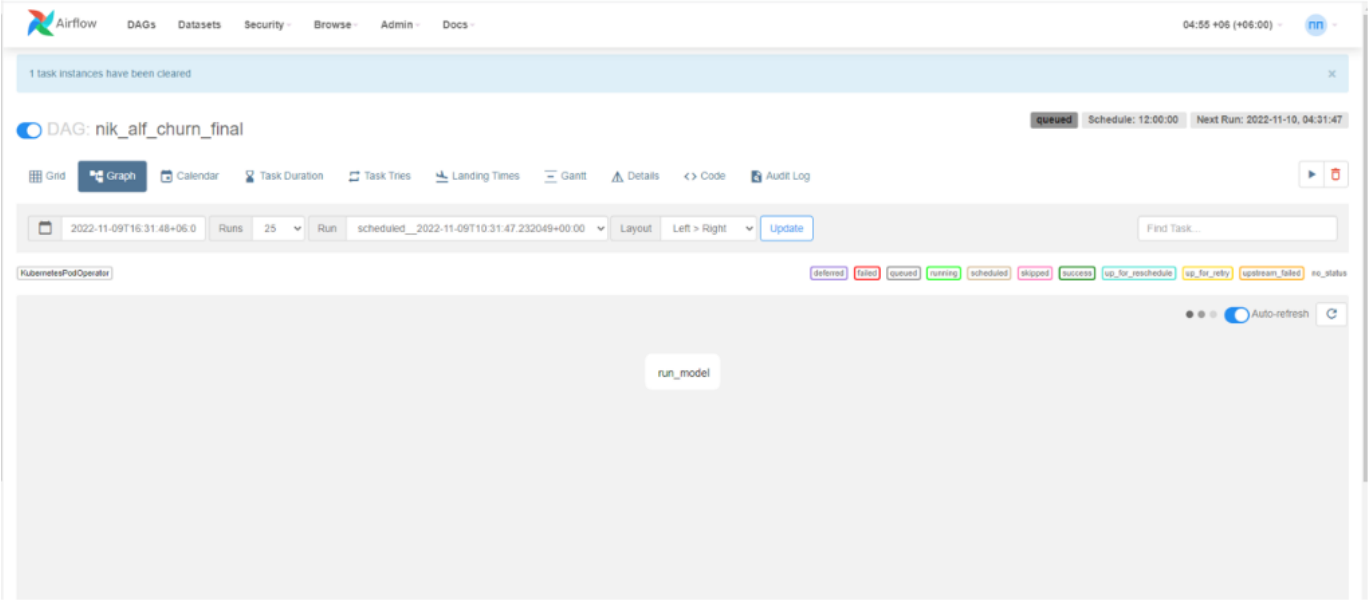
Под цифрой 9 указаны кнопка ручного запуска (не по расписанию, а прямо в данный момент) и кнопка удаления DAG-файла. Отметим, что при ручном запуске будет предложено две опции: “Trigger DAG” и “Trigger DAG w config”. В случае выбора первой, DAG-файл запустится как есть, в случае выбора второй опции, будет предложено заполнить json файл, где можно указать значение переменной, которое далее будет “проброшено” в контейнер с моделью как переменная окружения. Такой подход может быть полезен, когда нужно запустить скоринг моделью не по расписанию, а на конкретную дату.

Для активации или снятия с паузы нужного DAG-файла нужно, как было сказано выше, передвинуть ползунок в левом столбце. Далее можно “провалиться” в название данного файлы и попасть на страницу, изображенную на рисунке ниже:

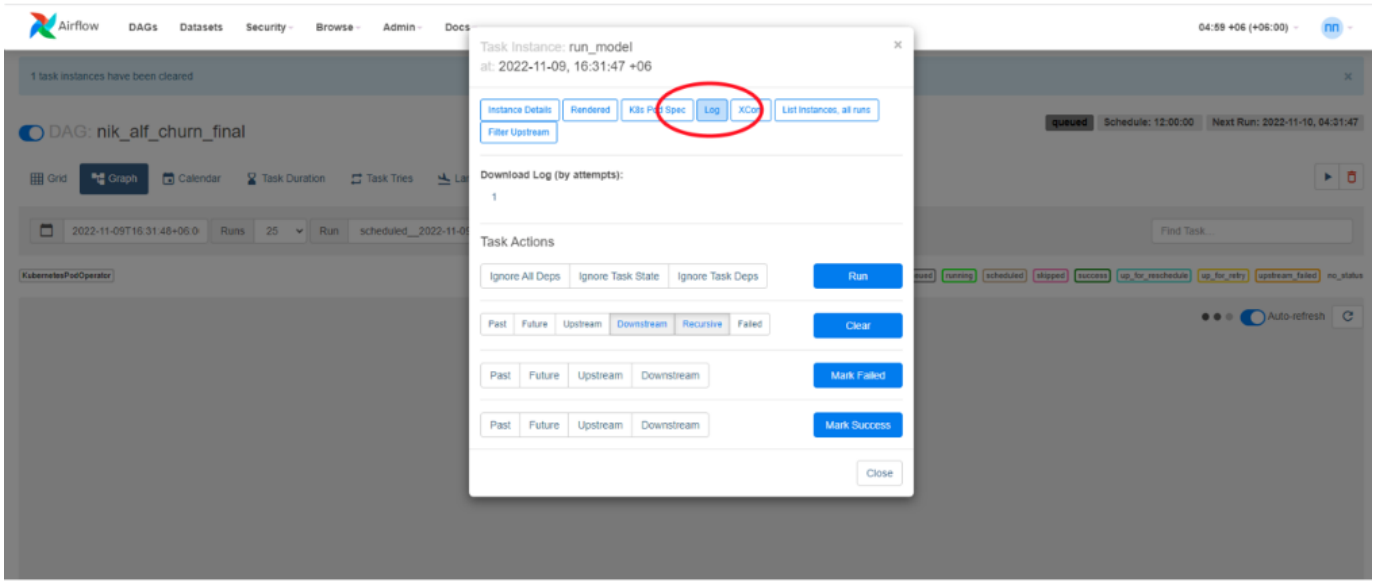


Далее нужно перейти на вкладку Graph (цифра 1), также стоит обратить на вкладку Code (цифра 2), нажав на которую можно перейти и увидеть код DAG-файла, что бывает полезно при отладке.

Перейдя на вкладку Graph на экране, будет прямоугольник “run_model” (для DAG-файла скоринга, не переобучения), рисунок «Интерфейс AirFlow2», можно отметить, что данный прямоугольник и является таской в данном DAG-файле.



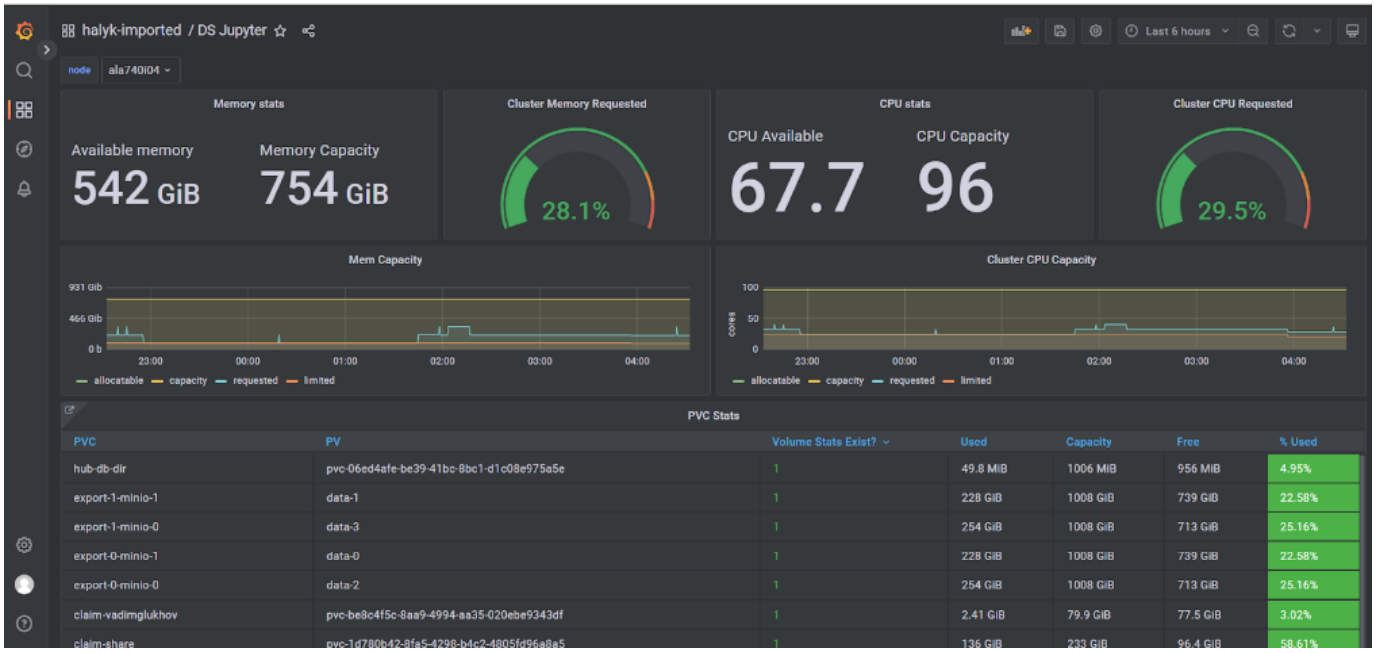
Затем нужно нажать левой кнопкой мыши на “run_model” и в появившемся окошке выбрать “Log”:



В открывшемся окне будут выводиться логи запуска данного DAG-файла, с помощью которых можно смотреть вывод лога модели. Работа в Airflow была рассмотрена на примере инстанса Airflow для среды DEV. Работа в Airflow-prod полностью аналогична.

Мониторинг системы

Для мониторинга технических параметров Системы используется инструмент Graphana, содержащий большое количество различных дашбордов. Для удобства Пользователя наиболее нужные показатели собраны по адресу **указать ссылку**, рисунок ниже:



Рассмотрим данный дашборд. В левой верхней части можно выбрать node, то есть узел кластера Kubernetes, для которого отображается мониторинг ресурсов. Следует отметить, что в кластере приложение (контейнер с Jupyter или контейнер с продуктивизированной моделью) может быть запущено только в том случае, если в кластере существует узел (node), имеющая достаточно ресурсов. Например, если Пользователь хочет запустить JupyterLab и выбирает 128Гб RAM, а в кластере отсутствует узел, на котором свободно 128гб RAM, то такой контейнер с Jupyter запуститься не сможет. Следует понимать, что если в описанном примере, когда Пользователь запрашивает 128Гб, а в кластере, например, есть два узла (node), на каждом из которых свободно 100Гб, то и в этом случае контейнер не сможет быть запущен, так как по-прежнему в кластере отсутствует узел с запрашиваемым набором ресурсов.

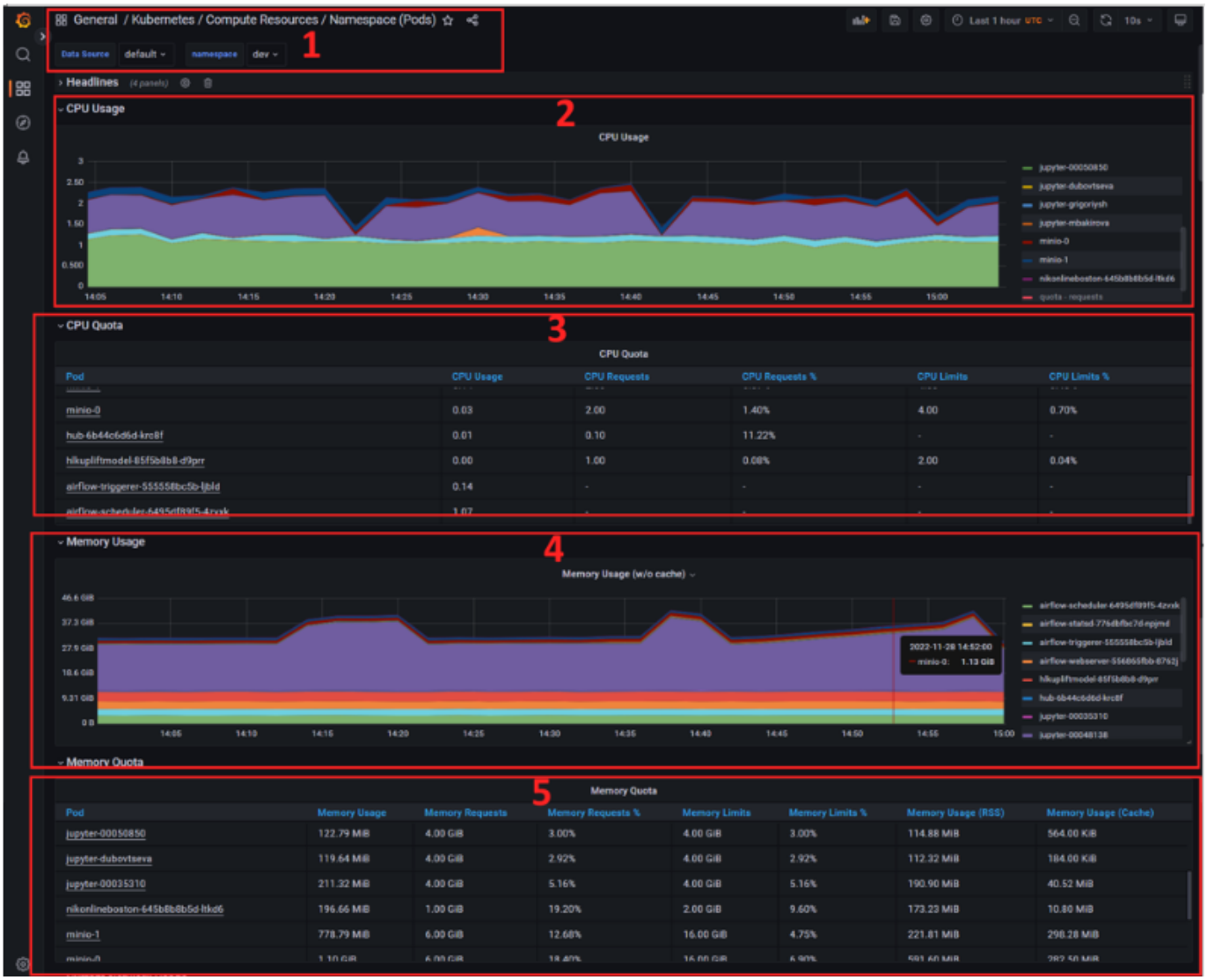
Ниже окна выбора node располагаются дашборды для мониторинга памяти RAM и CPU. Available memory показывает число свободных гигабайт оперативной памяти, Memory Capacity указывает объем оперативной памяти для данного узла, в полукруге показан процент зарезервированной (занятой) в данный момент памяти. Дашборд с CPU имеет аналогичные индикаторы, но показывает количество свободных и общих вычислительных ядер.

Еще ниже расположены графики потребляемых ресурсов как функция времени. Временной фрейм можно менять в меню, расположенном правом верхнем углу страницы.

Далее расположен дашборд PVC Stats, содержащий информацию об использовании файловой системы персональных персистивных хранилищ пользователей Системы (не путайте эту память с оперативной памятью данного узла!). В первом столбце таблицы указан ID (в формате claim-00012345) Пользователя, а в столбцах Used, Capacity, Free и %Used представлена информация об объеме используемой, общей, свободной и проценту используемой памяти соответственно. Следует отметить, что информация по файловой системе Пользователя доступна только, если он имеет открытый в данный момент JupyterLab.

Для расширенного мониторинга Пользователь может использовать и другие дашборды, одним из которых является просмотр запущенных pod'ов в данной среде (namespace), доступный по адресу:

Указать ссылку (рисунок ниже).



В первую очередь Пользователь должен выбрать namespace (dev или prod, цифра 1 на рисунке). Теперь на дашборде будет отображаться информация по pod'ам, имеющимся в данном неймспейсе. На цифре 2 отображается график используемых CPU как функция времени (масштаб таймфрейма можно выбрать в меню, расположенном справа вверху страницы), причем слева можно видеть названия pod'ов, которые потребляют ресурс. Названия вида jupyter-цифры_табельного_номера означают ресурсы, который использует Пользователь при работе с Jupyter. Также там будут отображаться названия моделей, которые в данный момент запущены в Airflow (для соответствующего контура). Именно запущены и работают в текущий момент, если модель стоит на регламенте, но в данный момент не запущена, то её не будет в данном списке. Стоит отметить, что в силу особенностей Kubernetes, имя модели может быть, во-первых, обрезано, а, во-вторых, содержать хэш из букв и цифр. В таблице ниже (цифра 3) дублируется информация с графика, но в формате таблице, где удобно смотреть реально используемые ("Usage") и запрошенные ("Requested") ресурсы. Также дашборд отображает информацию по RAM (цифры 4 и 5 на рисунке), логика отображения здесь полностью аналогичная информации о CPU, рассмотренная выше.

Разработка и продуктивизация онлайн моделей

Прежде чем разрабатывать онлайн модель, Пользователь должен быть хорошо знаком с Системой и иметь опыт разработки batch моделей.

Разработка онлайн моделей проводится с помощью создания проекта в Jupyter, важно напомнить, что имя проекта модели не должно содержать символа '_' и других символов, список которых выводится на экран при создании проект в ноутбуке create_project.ipynb.

Общая концепция разработки онлайн модели подразумевает, что Пользователь в Jupyter, например, в ноутбуке обучает модель, далее объект модели должен быть сохранен в папку pkl как model.pkl (обратите внимание, что файл должен быть сохранен в корневую папку pkl, а не папку online/src/pkl).

Необходимые для онлайн модели файлы расположены в папке online. Пользователь должен модифицировать файл в папке online/src/solver/main.py. Рассмотрим шаблон данного файла.

```
import pickle
import pandas as pd

def solve(temp_data):
    """
    Input: temp_data - json with features for prediction
    Output: f-string with result of prediction
    """
    with open("pkl/model.pkl","rb") as f:
        model = pickle.load(f)

    score = round(model.predict(pd.DataFrame([temp_data]))[0])

    return f'score={score}'
```

Файл содержит описание функции solve, которая на вход получает json-файл, содержащий наименования фичей и их значения для предсказания. Например, если модель должна предсказывать судьбу пассажира «Титаника» (известный учебный пример для Machine Learning), то данный json может представлять из себя данные по одному пассажиру, для которого планируется предсказание. Далее в файле должно происходить открытие файла pkl/model.pkl с обученной моделью. Далее должен быть код скоринга данной моделью входного json-файла. В строке return должен возвращаться результат скоринга.

Описанная в файле main.py функция solve вызывается в файле online/src/api.py, в котором описан код web-сервера онлайн модели, рассмотрим шаблон данного файла:

```
1       import os
2       import json
3       from fastapi import FastAPI, Body, status, UploadFile, File
4       from fastapi.responses import JSONResponse, FileResponse
5       from fastapi.exceptions import RequestValidationError
6       import uvicorn
7       from solver.main import solve
8       from scheme_api.scheme import ResponseData
9
10
11      config_api = {
12              "port": int(os.environ.get('PORT', '5005'))
13      }
14
15      app = FastAPI()
16
17
18      @app.exception_handler(RequestValidationError)
19      async def validation_exception_handler(request, exc):
20
21          response = {'http_status': 400, 'response_error': f'{exc}'}
22
23          return JSONResponse(content=response, status_code=400)
24
25
26      @app.get("/", status_code=status.HTTP_200_OK)
27      async def root():
28          return FileResponse("html/index.html")
29
30
31      @app.post("/solve", status_code=status.HTTP_200_OK, response_model=ResponseData)
32      async def solve_response(upload_file: UploadFile = File(...)):
33
34          if "json" in upload_file.filename:
35              data = json.load(upload_file.file)
36          else:
37              raise Exception("Invalid input!")
```

```
38
39          try:
40              response = solve(data)
41
42              return JSONResponse(status_code=200, content=response)
43
44          except Exception as e:
45              if 'error_response' in locals():
46                  error_response['error'] = str(e)
47              else:
48                  error_response = {'error': str(e)}
49
50              return JSONResponse(status_code=400, content=error_response)
51
52
53      if __name__ == "__main__":
54          uvicorn.run(app, host="0.0.0.0", port=config_api["port"])
```

На 7 строке происходит импорт функции solve из файла описанного выше файла main.py, далее на строке 40 результат вызова этой функции присваивается переменной response, которая далее участвует в выражении return на строке 42, содержащим ответ сервера, который будет показан на экране. В случае, если планируется переобучение онлайн модели, на этапе разработки модели в Jupyter необходимо подготовить файл retrain.py, находящийся в папке models (более подробно про файл retrain.py смотрите в разделе “Описание файла retrain.py”) и файл dag-ratrain.py из папки gitlab-ci, про который более подробно можно прочитать в разделе “Описание файла dag-retrain.py”.

Про выбор ресурсов контейнера продуктивизации онлайн модели следует смотреть раздел “Выбор ресурсов для продуктивизированного контейнера”.

Таким образом, при разработке онлайн модели Пользователь должен подготовить файл pkl/model.pkl и файл online/src/solver/main.py и, в случае переобучения, еще retrain.py и dag-retrain.py. Далее необходимо в терминале, запущенном в корневой директории проекта, выполнить команды:

```
dvc add ./pkl/model.pkl
```

```
dvc push
```

После этого нужно добавить на отслеживание в git все измененные файлы, например, с помощью команды в том же терминале:

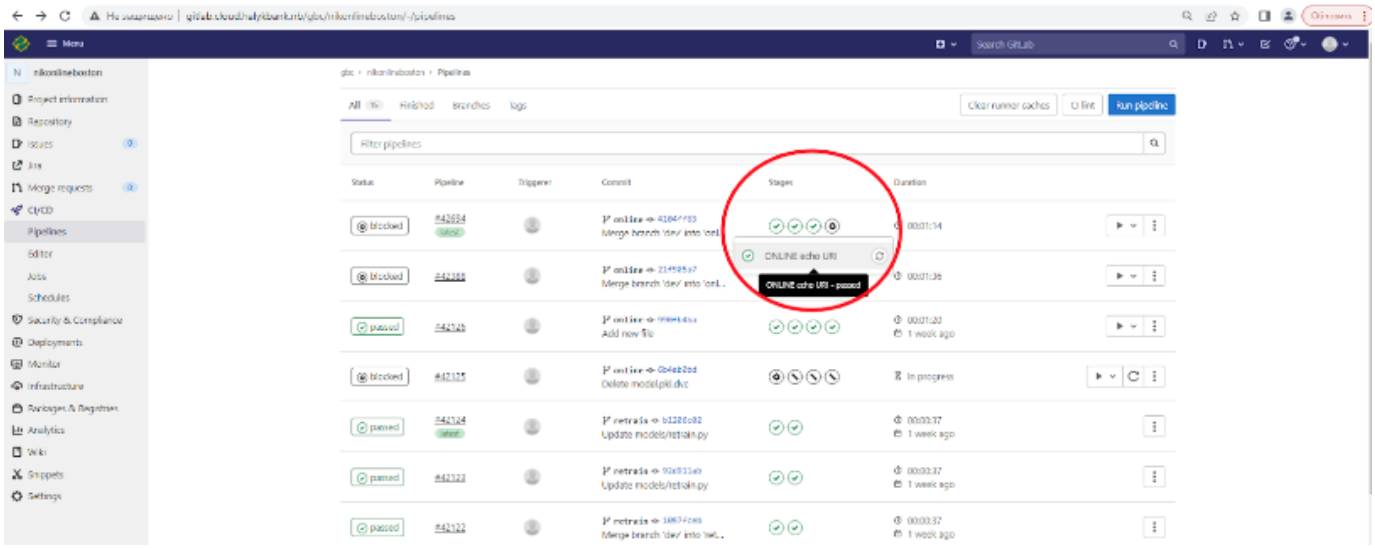
```
git add -A
```

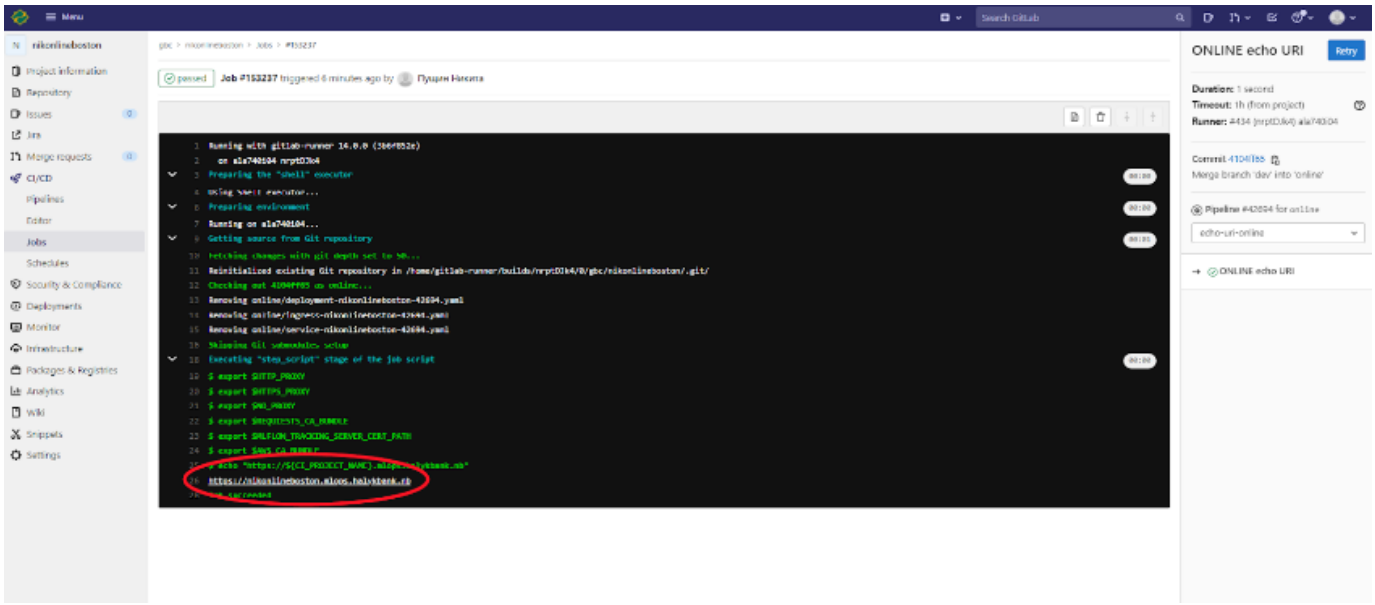
Далее выполнить команды:

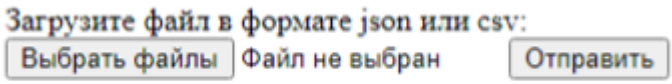
```
git commit -m “текст коммита”  git push origin dev
```

После этого нужно перейти в GitLab, найти свой проект и в ветке dev можно будет увидеть данный коммит. Рассмотрим процесс продуктивизации онлайн модели.

Далее необходимо сделать merge request из ветки dev в ветку online, данный merge request автоматически запустит CI/CD процесс продуктивизации модели. Перейти на вкладку CI/CD и запустить (кнопка “play” в меню справа). CI/CD процесс для онлайн модели состоит из четырех стадий. На первой стадии выполняется сборка образа с онлайн моделью, на второй стадии поднимается web-сервер, на третьей стадии в логах выводится адрес созданной онлайн модели. Чтобы увидеть лог файл стадии нужно нажать на иконку “круга” третьей стадии и кликнуть еще раз в появившееся окошко, на открывшейся странице лога внизу будет адрес, по которому можно перейти и в браузере откроется интерфейс модели (см. рисунки ниже).





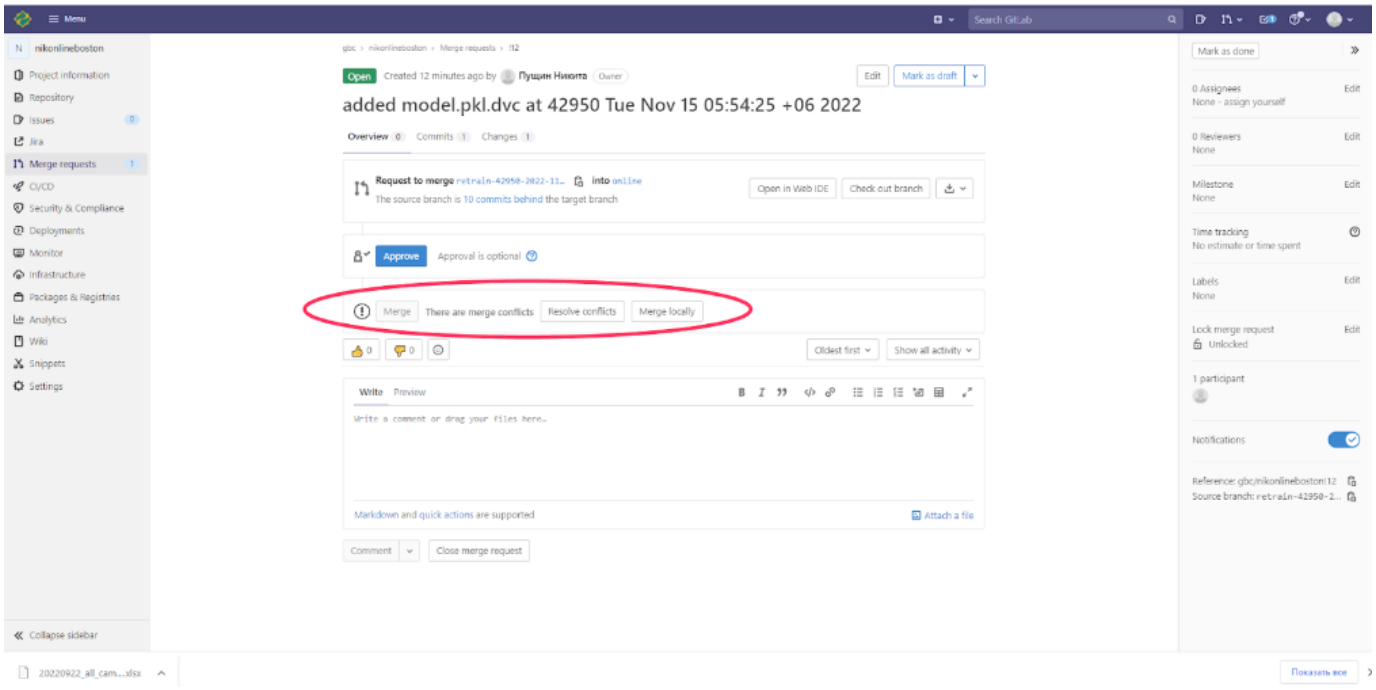


Далее необходимо нажать кнопку “Выбрать файлы”, после этого выбрать необходимый json-файл с фичами для предсказания и нажать кнопку отправить. После этого на экране будет показан ответ модели, а именно выражение, указанное в return функции solve из файла online/src/solve/main.py.

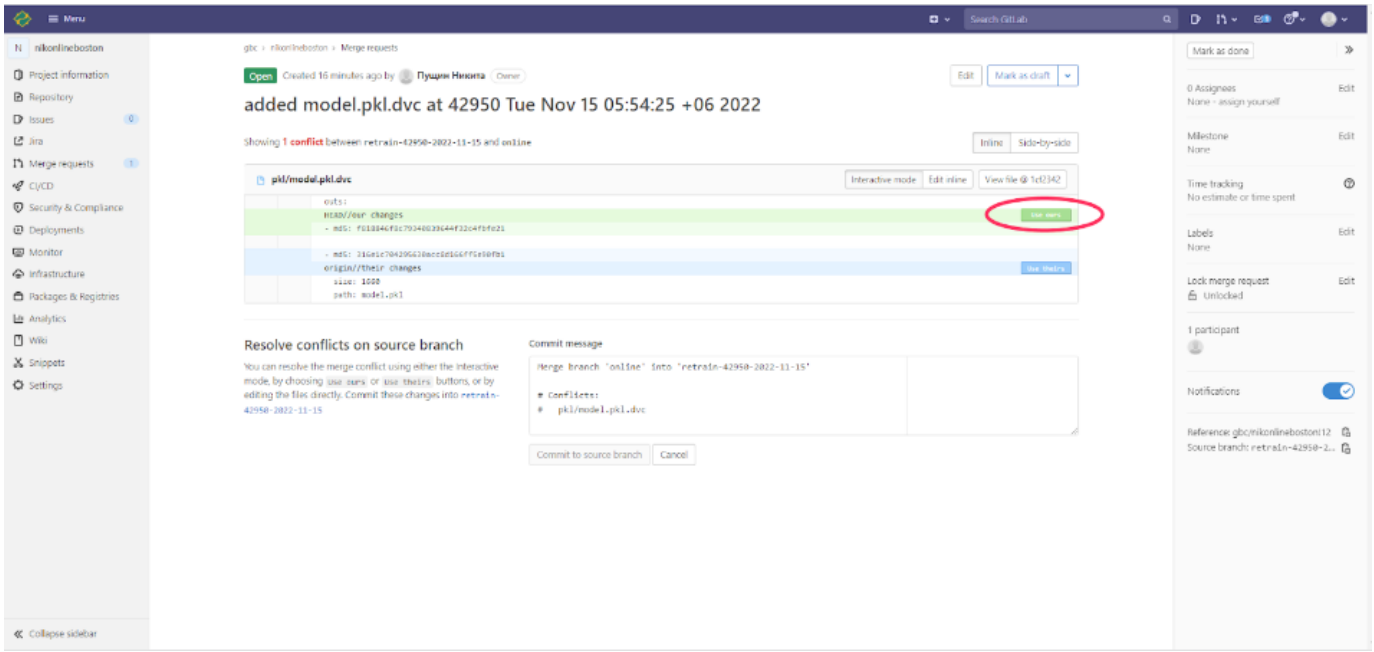
После того как работа с онлайн моделью завершена, нужно вернуться на вкладку CI/CD в GitLab и вручную закрыть web-сервер с моделью для освобождения ресурсов Системы. Для этого нужно навести курсор мышки на шестеренку (рисунок «Онлайн модель1») и запустить эту стадию. По окончанию работы CI/CD на последней стадии (когда последний “круг” закончит процесс и станет зеленым) web-сервер с онлайн моделью будет остановлен, а ресурсы Системы освобождены.

Для запуска переобучения онлайн модели и установки данного процесса на работу по расписанию необходимо выполнить слияние ветки dev в ветку retrain в репозитории GitLab. После этого в Airflow среды DEV по адресу **указать ссылку** появится DAG-файл, имеющий имя “имя_проекта_retrain”, который нужно активировать (более подробно про интерфейс Airflow смотрите в разделе “Оркестрация рабочих процессов машинного обучения”).

После работы данного DAG-файла в репозитории проекта модели в GitLab появится ветка с именем типа retrain-pipeline_id-date, в которой находится обновленный файл model.pkl.dvc. Далее необходимо выполнить слияние ветки retrain-pipeline_id-date в ветку online, при этом на этапе подтверждения может появиться уведомление о конфликтах в ветках.



В данном случае нужно нажать на “Resolve Conflict” и на открывшейся странице выбрать “Use ours” (рисунок 39) и далее нажать на появившуюся ниже кнопку “Commit to source branch”.



После успешного разрешения конфликта слияния должен начаться CI/CD процесс продуктивизации онлайн модели, который требует ручной активации. Для этого нужно перейти на вкладку CI/CD и запустить пайплайн подобно тому, как это было сделано при первой запуске онлайн модели.